

c o n f e r e n c e

.....
proceedings

**20th Large
Installation System
Administration
Conference**

Washington, D.C., USA

December 3–8, 2006

Sponsored by
The **USENIX Association** and **SAGE**

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

THE USENIX SIG FOR
[sage]
SYSADMINS

For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: +1 510-528-8649
<http://www.usenix.org>
<office@usenix.org>

Past LISA Conferences

LISA '05: 19th Large Installation System Administration Conference	Dec. 4-9, 2005	San Diego, CA
LISA '04: 18th Large Installation System Administration Conference	Nov. 14-19, 2004	Atlanta, GA
LISA '03: 17th Large Installation Systems Administration Conference	Oct. 26-31, 2003	San Diego, CA
LISA '02: 16th Systems Administration Conference	Nov. 3-8, 2002	Philadelphia, PA
LISA 2001: 15th Systems Administration Conference	Dec. 2-7, 2001	San Diego, CA
LISA 2000: 14th Systems Administration Conference	Dec. 3-8, 2000	New Orleans, LA
LISA '99: 13th Systems Administration Conference	Nov. 7-12, 1999	Seattle, WA
LISA '98: 12th Systems Administration Conference	Dec. 6-11, 1998	Boston, MA
LISA '97: 11th Systems Administration Conference	Oct. 26-31, 1997	San Diego, CA
LISA '96: 10th System Administration Conference	Sept. 29-Oct. 4, 1996	Chicago, IL
LISA '95: 9th System Administration Conference	Sept. 18-22, 1995	Monterey, CA
LISA '94: 8th USENIX System Administration Conference	Sept. 19-23, 1994	San Diego, CA
LISA '93: USENIX 7th System Administration Conference	Nov. 1-5, 1993	Monterey, CA
LISA VI: 6th Systems Administration Conference	Oct. 19-23, 1992	Long Beach, CA
5th Large Installation Systems Administration Conference	Sept. 30-Oct. 3, 1991	San Diego, CA
4th Large Installation System Administrator's Conference	Oct. 17-19, 1990	Colorado Springs, CO
Workshop on Large Installation Systems Administration III	Sept. 7-8, 1989	Austin, TX
Workshop on Large Installation Systems Administration	Nov. 17-18, 1988	Monterey, CA
Workshop on Large Installation Systems Administration	April 9-10, 1987	Philadelphia, PA

Copyright © 2006 by The USENIX Association. All rights reserved.
This volume is published as a collective work.
Rights to individual papers remain with the author or the author's employer.
Permission is granted for the noncommercial reproduction
of the complete work for educational or research purposes.

USENIX acknowledges all trademarks appearing herein.

ISBN 1-931971-48-X

USENIX Association

**Proceedings of the
20th Large Installation
System Administration Conference
(LISA '06)**

**December 3-8, 2006
Washington, DC, USA**

ACKNOWLEDGMENTS

PROGRAM CHAIR

William LeFebvre, *Independent Consultant*

PROGRAM COMMITTEE

Narayan Desai, *Argonne National Laboratory*

Peter Galvin, *Corporate Technologies, Inc.*

Trey Harris, *Amazon.com*

John “Rowan” Littell, *California College
of the Arts*

Adam Moskowitz, *Menlo Computing*

Mario Obejas, *Raytheon*

Tom Perrine, *Sony Computer*

Entertainment America

W. Curtis Preston, *GlassHouse Technologies*

Amy Rich, *Tufts University*

Marc Staveley, *SOMA Networks Inc.*

Rudi Van Drunen, *Leiden Cytology
and Pathology Labs*

Alexios Zavras, *IT Consultant*

INVITED TALKS COMMITTEE

David Blank-Edelman, *Northeastern
University CCIS*

Doug Hughes, *Global Crossing*

GURU IS IN COORDINATOR

Philip Kizer, *Estacado Systems*

WORK-IN-PROGRESS REPORTS COORDINATOR

Esther Filderman, *Pittsburgh Supercomputing
Center*

NATIVE GUIDES

Jennifer Ash-Poole, *AdNet/GSFC*

Laura Carriere, *SAIC/GSFC*

WORKSHOPS COORDINATOR

Luke Kanies, *Reductive Labs*

ADVANCED TOPICS WORKSHOP

Adam Moskowitz, *Menlo Computing*

CONFIGURATION MANAGEMENT WORKSHOP

Paul Anderson, *University of Edinburgh*

DATACENTER MANAGEMENT WORKSHOP

Robert Sidney Wilroy, *George Mason
University*

MANAGING SYSADMINS WORKSHOP

Tom Limoncelli, *Google, Inc.*

Cat Okita, *Earthworks*

CONFIGURATION MGMT TOOLS & PRACTICE WORKSHOP

Luke Kanies, *Reductive Labs*

Narayan Desai, *Argonne National Laboratory*

SOFTWARE LICENSING WORKSHOP

Sandra Bittner, *Argonne National Laboratory*

UNIVERSITY ISSUES WORKSHOP

John “Rowan” Littell, *California College
of the Arts*

PROCEEDINGS TYPESETTING

Rob Kolstad

CONFERENCE ORGANIZATION

The USENIX Association Staff

EXTERNAL REVIEWERS

Alva Couch

Esther Filderman

Tim Hunter

Tom Limoncelli

Cat Okita

David Snyder

David Williamson

CONTENTS

Acknowledgments	ii
Index of Authors	vi
Message from the Program Chair	vii

WEDNESDAY, DECEMBER 6

Opening Remarks and Keynote

Session Chair: William LeFebvre

Hollywood's Secret War on Your NOC

Cory Doctorow – Boing Boing

Electronic Mail

Session Chair: John “Rowan” Littell

1	Privilege Messaging: An Authorization Framework over Email Infrastructure <i>Brent ByungHoon Kang, Gautam Singaraju, and Sumeet Jain – University of North Carolina at Charlotte</i>
17	Securing Electronic Mail on the National Research and Academic Network of Italy <i>Roberto Cecchini – INFN, Florence; Fulvia Costa – INFN, Padua; Alberto D'Ambrosio – INFN, Turin; Domenico Diacono – INFN, Bari; Giacomo Fazio – INAF, Palermo; Antonio Forte – INFN, Rome; Matteo Genghini – IASF, Bologna; Michele Michelotto – INFN, Padua; Ombretta Pinazza – INFN, Bologna; Alfonso Sparano – University of Salerno</i>
31	A Forensic Analysis of a Distributed Two-Stage Web-Based Spam Attack <i>Daniel V. Klein – LoneWolf Systems</i>

Boundaries

Session Chair: Rudi van Drunen

41	Firewall Analysis with Policy-Based Host Classification <i>Robert Marmorstein and Phil Kearns – The College of William and Mary</i>
53	Secure Mobile Code Execution Service <i>Lap-chung Lam, Yang Yu, and Tzi-cker Chiueh – Rether Networks, Inc.</i>
63	FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs <i>Adam Slagell, Kiran Lakkaraju, and Katherine Luo – NCSA, University of Illinois at Urbana-Champaign</i>

Security

Session Chair: Adam Moskowitz

79	Centralized Security Policy Support for Virtual Machine <i>Nguyen Anh Quynh, Ruo Ando, and Yoshiyasu Takefuji – Keio University</i>
89	A Platform for RFID Security and Privacy Administration <i>Melanie R. Rieback – Vrije Universiteit, Amsterdam; Georgi N. Gaydadjiev – Delft University of Technology; Bruno Crispo, Rutger F. H. Hofman, and Andrew S. Tanenbaum – Vrije Universiteit, Amsterdam</i>

THURSDAY, DECEMBER 7

Theory

Session Chair: Narayan Desai

- 103 Specification-Enhanced Policies for Automated Management of Changes in IT Systems**
Chetan Shankar – University of Illinois at Urbana-Champaign; Vanish Talwar, Subu Iyer, Yuan Chen, and Dejan Milojicic – Hewlett-Packard Laboratories; Roy Campbell – University of Illinois at Urbana-Champaign
- 119 Experience Implementing an IP Address Closure**
Ning Wu and Alva Couch – Computer Science Department, Tufts University
- 131 Modeling Next Generation Configuration Management Tools**
Mark Burgess – Oslo University College; Alva Couch – Tufts University

Analysis

Session Chair: Mario Obejas

- 149 Windows XP Kernel Crash Analysis**
Archana Ganapathi, Viji Ganapathi, and David Patterson – University of California, Berkeley
- 161 SUEZ: A Distributed Safe Execution Environment for System Administration Trials**
Doo San Sim and V. N. Venkatakrishnan – University of Illinois, Chicago
- 175 WinResMon: A Tool for Discovering Software Dependencies, Configuration and Requirements in Microsoft Windows**
Rajiv Ramnath – National University of Singapore; Sufatrio – Temasek Laboratories, National University of Singapore; Roland H. C. Yap and Wu Yongzheng – National University of Singapore

Systems and Network Management

Session Chair: Elizabeth Zwicky

- 187 LiveOps: Systems Management as a Service**
Chad Verbowski – Microsoft Research; Juhan Lee and Xiaogang Liu – Microsoft MSN; Roussi Roussev – Florida Institute of Technology; Yi-Min Wang – Microsoft Research
- 205 Managing Large Networks of Virtual Machines**
Kyrre M Begnum – Oslo University College, Norway
- 215 Directing Change Using Bcfg2**
Narayan Desai, Rick Bradshaw, Joey Hagedorn, and Cory Lueninghoener – Argonne National Laboratory

Visualization

Session Chair: Amy Rich

- 221 NAF: The NetSA Aggregated Flow Tool Suite**
Brian Trammell – CERT/NetSA, Carnegie Mellon University; Carrie Gates – CA Labs
- 233 Interactive Network Management Visualization with SVG and AJAX**
Athanasios Douitsis and Dimitrios Kalogeras – National Technical University of Athens, Greece
- 247 Bridging the Host-Network Divide: Survey, Taxonomy, and Solution**
Glenn A. Fink and Vyas Duggirala – Virginia Polytechnic Institute and State University; Ricardo Correa – University of Pennsylvania; Chris North – Virginia Polytechnic Institute and State University

FRIDAY, DECEMBER 8

Potpourri

Session Chair: David Parter

- | | |
|-----|--|
| 263 | The NMI Build & Test Laboratory: Continuous Integration Framework for Distributed Computing Software
<i>Andrew Pavlo, Peter Couvares, Rebekah Gietzel, Anatoly Karp, Ian D. Alderman, and Miron Livny – University of Wisconsin-Madison; Charles Bacon – Argonne National Laboratory</i> |
| 275 | Unifying Unified Voice Messaging
<i>Jon Finke – Rensselaer Polytechnic Institute</i> |
| 287 | Fighting Institutional Memory Loss: The Trackle Integrated Issue and Solution Tracking System
<i>Daniel S. Crosta and Matthew J. Singleton – Swarthmore College Computer Society; Benjamin A. Kuperman – Swarthmore College</i> |

INDEX OF AUTHORS

Ian D. Alderman	263	Daniel V. Klein	31
Ruo Ando	79	Benjamin A. Kuperman	287
Charles Bacon	263	Kiran Lakkaraju	63
Kyrre M Begnum	205	Lap-chung Lam	53
Rick Bradshaw	215	Juhan Lee	187
Mark Burgess	131	Xiaogang Liu	187
Roy Campbell	103	Miron Livny	263
Roberto Cecchini	17	Cory Lueninghoener	215
Yuan Chen	103	Katherine Luo	63
Tzi-cker Chiueh	53	Robert Marmorstein	41
Ricardo Correa	247	Michele Michelotto	17
Fulvia Costa	17	Dejan Milojicic	103
Alva Couch	119, 131	Chris North	247
Peter Couvares	263	David Patterson	149
Bruno Crispo	89	Andrew Pavlo	263
Daniel S. Crosta	287	Ombretta Pinazza	17
Alberto D'Ambrosio	17	Nguyen Anh Quynh	79
Narayan Desai	215	Rajiv Ramnath	175
Domenico Diacono	17	Melanie R. Rieback	89
Athanasios Douitsis	233	Roussi Roussev	187
Vyas Duggirala	247	Chetan Shankar	103
Giacomo Fazio	17	Doo San Sim	161
Glenn A. Fink	247	Gautam Singaraju	1
Jon Finke	275	Matthew J. Singleton	287
Antonio Forte	17	Adam Slagell	63
Archana Ganapathi	149	Alfonso Sparano	17
Viji Ganapathi	149	Sufatrio	175
Carrie Gates	221	Yoshiyasu Takefuji	79
Georgi N. Gaydadjiev	89	Vanish Talwar	103
Matteo Genghini	17	Andrew S. Tanenbaum	89
Rebekah Gietzel	263	Brian Trammell	221
Joey Hagedorn	215	V. N. Venkatakrishnan	161
Rutger F. H. Hofman	89	Chad Verbowski	187
Subu Iyer	103	Yi-Min Wang	187
Sumeet Jain	1	Ning Wu	119
Dimitrios Kalogeras	233	Roland H. C. Yap	175
Brent ByungHoon Kang	1	Wu Yongzheng	175
Anatoly Karp	263	Yang Yu	53
Phil Kearns	41		

Message from the Program Chair

Dear Reader:

In your hands (or on your screen) is a book containing the most significant and important research and development work done this year in system administration. It is my pleasure as program chair to present this body of work to you.

It is only through the tireless effort of many individuals that I am able to provide you with this body of work. This effort was put forth by the authors themselves, their advisors, the program committee, the external reviewers, the shepherds, the USENIX staff, and of course our typesetting master, Rob Kolstad. I am grateful to each and every one of them for all of their excellent work.

I believe very strongly in the advancement of our profession through the publication of peer-reviewed research. LISA is one of the very few venues where people can publish new work that strives for the advancement of system administration as a profession and a discipline. Through this work we can all benefit. It is very important to me personally that our profession have this outlet, and I am very proud to be a part of the process this year.

These Proceedings from the 2006 LISA conference contain the complete texts of the refereed papers. Of the 49 papers that were submitted, 23 are presented here. The selection process is always a challenge, and many good papers could not be accepted. We always encourage those who were turned down to resubmit in future years.

For 20 years, LISA has been the best conference for system administrators. The success of these conferences was no accident. It came about because of the efforts of many individuals. I would like to thank everyone who has had a part in presenting LISA through the years: authors, speakers, tutorial instructors, gurus, past program chairs, committee members, co-ordinators, attendees, hotel and A/V staff, and especially the USENIX staff. This is a team effort and everyone has contributed to the success of LISA. I hope you enjoy the fruits of their labor.

William LeFebvre
Program Chair
LISA '06

Privilege Messaging: An Authorization Framework over Email Infrastructure

Brent ByungHoon Kang, Gautam Singaraju, and Sumeet Jain
– University of North Carolina at Charlotte

ABSTRACT

The current email infrastructure is burdened by multiple resource constraints and a plethora of security issues. Apart from the fact that email users are spending more time and effort sifting through unsolicited emails, more serious problems such as Phishing are on the rise. This can be attributed to a fundamental shortcoming in the current email infrastructure: a lack of an authorization framework. This allows any user to create content in anyone's mailbox. In this paper, we revisit the fundamental problem of non-existent authorization and discuss the design of an effective authorization service overlaying the existing email infrastructure. We propose Privilege Messaging (P-Messaging), a fine-granular authorization framework that operates on the principle that a sender requires a set of privileges in order to send messages, simultaneously enables the receiver's infrastructure server to verify the messages before accepting it. We present a prototype implementation and discuss its benefits. An automatic classification of email can be effectively performed based on the privilege-tag. Privilege-tag can provide flexible and fine-granular reputation management than current domain-based solutions. The use of privilege-tag as entry ID in a white-list can be more manageable than the use of individual email address. Finally, the privilege-tag can be used as an email header, retaining the benefits of currently deployed MTA architecture, namely reliability and flexibility.

Introduction

Email, a simple and cost effective messaging technology, has become the universal mode of interaction over the Internet. Undeniably, email is so established that day to day commercial and non-commercial activities are contingent upon its reliable operation. However, the Internet explosion has also introduced a variety of new risks and threats. Unsolicited email has reached epidemic proportions, severely limiting the usability of the current email infrastructure with non-essential resource consumption.

Spam, interchangeably referred to as Unsolicited Bulk Email and/or Unsolicited Commercial Email, is just one of the key threats faced by the current email infrastructure. Conveying worthless or fraudulent information, spam adversely affects the recipient in terms of time and money by encumbering limited resources such as server space [7]. Apart from the end user, each of the numerous relay stations also pays toward the resources for routing, bandwidth and memory.

Another threat, Phishing [13], fools recipients into divulging their financial information by redirecting them to a masqueraded site. The US Electronic Payments Association estimated that world wide financial losses due to Phishing reached approximately \$500 million in 2004 [14].

In addition to the Spam and Phishing problem existent in the current email infrastructure, users spend significant amounts of their time and money classifying and labeling their emails. There has been only limited automatic classification mechanisms based on user-

specified rules. In other words, there does not exist a Quality of Service (QoS) mechanism for users to classify emails based upon the email's relevance and importance. The classification and the relevance of email should be based on associated trust information, rather than a classification based on its content and properties.

Recent legislations, such as CAN-SPAM [4], introduced to curtail unsolicited emails, have been unsuccessful due to the impracticality of monitoring large numbers of email communications all over the world. In a desperate effort to curtail spam, some organizations have undertaken measures such as maintaining an isolated internal email infrastructure. Other prevalent techniques include the use of spam filters or the blacklisting of email accounts. These systems provide an excellent mechanism to weed out most unsolicited emails; however, they might, potentially, blacklist a legitimate email account, rendering it ineffective. Thus, the financial losses to organizations are not only due to the unsolicited emails but also from legitimate emails being falsely classified as unsolicited email [6].

The fundamental reason for the threats faced by the current email infrastructure comes from the absence of an authorization framework. Currently, any user on the Internet can send messages to another user's mailbox with no mechanism that differentiates between authorized and unauthorized senders. This is the primary reason for the proliferation of both illegal and uninvited content. There is an obvious need for a system that allows only authorized emails to be accepted by the receiver.

We propose a Privilege Messaging (P-Messaging) Framework, which enables a legitimate sender to send email and the receiver's server to verify the email's privileges before the content is created in the receiver's mailbox. A privilege can be viewed as a credential associated with a group of users. An email can hold multiple privileges, for instance, one as a faculty member and another as a USENIX member. Furthermore, P-Messaging provides trust-based messaging, given that under the P-Messaging framework each email is signed (i.e., vouched as to its validity) by the sender's P-Messaging infrastructure server. The trust-based mechanism supports the authorization mechanism through the creation and maintenance of a Circle of Trust among the P-Messaging providers.

The rest of the paper is organized as follows: the next section examines the related work in this area. Subsequent sections detail the design and architecture of P-Messaging, describe the benefits of P-Messaging, and discuss usage scenarios. Next is the evaluation of P-Messaging, future work, and finally the conclusion.

Related Work

Network-based email communication has existed since the early days of ARPANET [9] enabling a small, close-knit group to communicate electronically. Today, even with the extensive usage of email infrastructure [7], there exists no authorization service; hence, there is no way to guard against fraudulent mailers sending unsolicited messages to another user. To combat this, multiple filtering technologies have been developed that weed out most, but not all of the unsolicited email.

Figure 1 shows the comparison between the domain-based solutions and the user based white-list maintenance. Domain based solutions have credential information of the mail servers in their DNS entries; the credentials that are available are varied and dependent upon the solution adopted. The advantage of Domain based solution is that these systems allow trust based communication among the sender and the receiver. However, these publicly available credentials

are not validated by a common trusted third party. Spammers and Phishers have setup their own domains and have continued to spam. Domain based solutions provide little QoS: we define QoS as automatic classification of the emails based on both the sender credentials associated with the email and the credentials that are accepted by the receiver.

In case of the white-list based solutions, the email classification is performed by checking the sender's email IDs against the white-list maintained by the receiver. A new correspondent might be considered an unsolicited sender unless the email ID is enlisted into the white-list beforehand. The white-list size can grow unbounded because each user needs to list all the email IDs that they deem valid. Moreover, if the email ID needs to be revoked or changed, it requires all the user's contacts to update their white-lists.

These systems can be classified based on a sender's information and/or analyzing the email's contents. We categorize them into two categories that scrutinize the message based on a) the characteristics of the email content and blacklisted email IDs; and b) the sender domain's credential.

Classification Based on Email's Content and Collaborative Blacklisting

Word filters [1] search for patterns and remove the most obvious spam; however, spammers have often circumvented word filters by using misspelled words. Thus, word filters requires regular updates with commonly misspelled words in unsolicited emails. Rule-Based scoring mechanisms check for keywords, but use rules to analyze emails. For instance, depending on the score received by a particular email, it is classified as either spam or not spam. Bayesian Filters, on the other hand, perform lexicographical and statistical analysis on the email for words and/or phrases depending on the recipient's previous emails.

Incorporating user feedback at the MTA level forms the basis of another technique: collaborative filters [8]. With collaborative filters, an unsolicited email is filtered at the MTA with users' feedback about

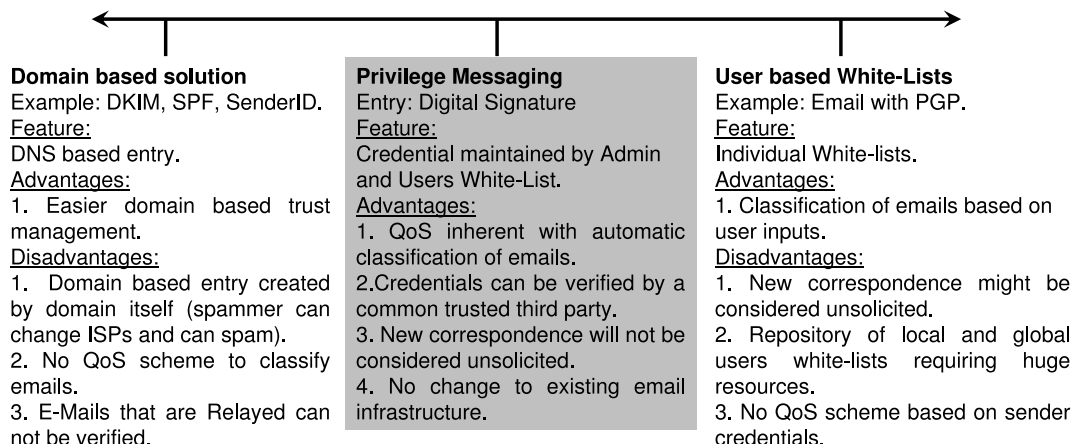


Figure 1: Comparison of Privilege Messaging with current technologies.

falsely classified emails, hence developing a smarter spam filter. However, an email that is marked as spam by a user might not be considered spam by another user.

A combination of different techniques provides a reliable means to classify an email as spam [11]. SpamGuru [18] employs multiple techniques such as word filters and Chung-Kwei. Chung-Kwei uses a pattern discovery technique, to classify unsolicited emails.

Other techniques exist such as HoneySpam that borrows the idea of honeypots to email infrastructure, the Social Network based classification, and a new email delivery framework that proposes receiver-pull instead of sender-push [3, 5, 15]. However, these systems do not address the essential problem of unrestricted access to others' mailboxes.

Classification Based on Sender Credential

Blacklist IP is comparatively simpler and computationally less intensive than other techniques. This process keeps a list of IP addresses identified as spammers and another white-list for legitimate users. Real-time Blackhole List (RBL) [17] works similar to Blacklist IP, but RBLs are not manually updated by individual organizations. Instead, RBL operators maintain public RBLs to which organizations subscribe.

PGP [16, 22] and SenderID [12] techniques allow verification of a sender's email address based on the sender's domain credentials. PGP is a fine-granular service containing a list of individual users' public keys. User contact management based on PGP keys can provide the benefits of identifying trusted peer correspondents as well as verifying the email integrity. However, utilizing PGP-based authentication techniques incurs the overhead of requiring individual users to maintain such lists. A new correspondence might be considered unsolicited, unless the email ID is enlisted beforehand. Also, the size of the white-list can grow unbounded because the local and global white-lists may need to list all the legitimate email IDs on the Internet.

SenderID addresses the problem of spamming and Phishing by validating an email's origin, i.e., by verifying the IP address presented by the email against the sending domain. This validation is performed by comparing the sender's IP address against the registered domain's mail servers. SenderID is a coarse-granular service validating a sender's domain. By coarse-granular, we mean that there is only one credential available for the entire domain.

Sender Policy Framework (SPF) [20] is a technique that has been introduced to prevent email forgery. In SPF, the mail servers are identified by DNS entries which receivers use to validate the sender's MTA. The DNS records also indicate the sender's adopted policy, for instance, the list of mail servers allowed to send email from a domain. At the receiver's end, when the mail is received, the receiver checks the sender's policy specified through their DNS records. For example, if the sender's email

server is not the one specified in the policy, the email is considered unsolicited.

Domain Key Identified Mail (DKIM) [2] attempts to reduce the traffic on the network by publishing the mail server's public keys through DNS records. Each domain creates and publishes its public key. Each email sent henceforth is signed by the sender's mail server. The receiver's mail server can verify the digital signature by retrieving the public key of the sender from their DNS records. Thus, the sender's domain information, as presented by the email, is validated with the actual domain. However, in DKIM, the public-private key pair is generated by the domain itself. Additional security can be provided by publishing the keys at a Certificate Authority (CA) [21]. However, presently DKIM does not enforce this restriction.

Although there are major advantages with these systems that classify emails based on sender credentials, there exist distinct disadvantages. Firstly, a significant portion of spam is usually sent from perfectly legitimate domains. For instance, the spammer can spam by creating a new domain with the required credentials, allowing them to send spam. Secondly, relaying emails cannot be performed because the sender's domain information in the email will be different from that of the relaying domain.

These systems motivate a need for a fine-granular service: finer than domain-based solutions and coarser than white-list management. A coarse-granular service removes the need to maintain a local white-list, whereas a fine-granular service allows each email to be associated with multiple privileges rather than just one as in the domain-based solution. Using a fine-granular service also allows a negative reputation to be restricted within a small group of users rather than scaling to the complete domain. Hence, there is a need for a framework that allows users to explicitly specify which group of email IDs has the appropriate credentials to create content in their own mailbox. In other words, an authorization framework is required that verifies the sender and based on the authorization presented, be able to classify mails for them. In order to provide such a scalable authorization framework, we introduce Privilege Messaging.

Privilege Messaging

The systems described in the previous section are some of the techniques that the current email infrastructure has adopted in order to classify emails as spam and legitimate emails. These systems identify spam based on either the email contents or the sender domain. However, they do not provide authorization services before the content is created in receiver's mailbox. Moreover, a domain is responsible for the creation and maintenance of its own authentication credentials. Though such a technique considerably restricts a spammer by requiring extensive computation before sending

a bulk email, such a mechanism does not allow for qualified trust to be placed by the receiver.

Revisiting the fundamental issue of non-existent authorization problem is a first step toward designing a system that can be an effective solution to the security loopholes in the email infrastructure. Furthermore, any attempt to provide a better email infrastructure should address this non-existent authorization. To combat proliferation of unwanted traffic using an authorization service, a new system must be designed with a scalable architecture to enable fine-granular control for sending and verification of emails.

Thus, the need for P-Messaging, a system that provides authorization, cannot be emphasized more. P-Messaging stipulates that the verification of the privileges held by the email will determine its authenticity and relevance to a particular user. Each email ID can be assigned multiple privileges. For instance, each email ID would be associated with various groups, such as one for each department in a school or one for each project team in a development environment. The granularity of a group can be defined by the users. Using P-Messaging, an email ID, based on the privileges held, can send and receive email with authorized users with the help of a privilege. As each privilege has a PKI key pair, digital signatures are used to verify the privileges associated with the emails.

Capable of supporting different Message Transfer Agents (MTA), P-Messaging provides both better security and improved QoS over the present email infrastructure. The MTAs provide the essential services of relaying, spooling, queuing and sending emails; P-Messaging is a gradual process of introducing the authorization feature. To support such deployment, P-Messaging framework along with the MTAs provides trust based communications.

In most cases, unsolicited email arises from mail servers that are either (i) Zombie or (ii) Unauthorized servers set up temporarily. Towards controlling the unwanted traffic from them, creation and maintenance of a Circle of Trust (CoT) among P-Messaging providers is essential. A CoT is a trust relationship formed between a sender and a receiver due to the trust placed on them by a common third party entity, such as a Certificate Authority (CA). Using CoT, qualified trust can be placed by a receiver on a sender since the CA is responsible for maintaining the trust among the servers. Each of these servers in turn maintains a level of trust on each privilege they maintain.

With the help of the CoT, any email received that is not trusted is placed in a no-privilege class, which forms the lowest available privilege class. P-Messaging classifies emails for the receivers into multiple classes; only emails from the members of CoT could be classified. Any attempt to create unsolicited content will result in the trust being revoked. Members of the CoT are capable of informing the admin of a particular system about possible spam arising from it. Hence, to

be able to use Privilege Messaging, each server in the CoT should continually strive to remain in the CoT. Thus, the CoT creates trust among P-Messaging Providers. The trust maintenance among the members will be discussed later.

The following sections discuss the functional components of P-Messaging, architecture for CoT, followed by the architectures of P-Messaging and finally management of the P-Messaging providers.

P-Messaging: Functional Components

This subsection discusses the functional components of P-Messaging. These components consist of the:

1. P-Messaging Server
2. P-Messaging Privilege Verifier
3. P-Messaging Manager
4. P-Messaging Trust Authority

P-Messaging Server

Architecturally, P-Messaging Server (P-Server) is a sandbox before the MTA. To send an email, the users interact with the P-Server, which in turn interacts with the MTA. After receiving an email from a user, the P-Server validates the user and attaches the credentials (i.e., privileges) to the message. Finally, the message along with the credentials is transmitted through the MTA.

The P-Server provides different services: (i) User Authentication Service, (ii) Privilege Lookup Service, (iii) Message Signing Service and (iv) Privilege Admin Service. The User Authentication Service provides a mechanism to authenticate a user to the P-Server. This Authentication Service can be password-based authentication scheme. Privilege Lookup Service, based on a rule-based engine, allows senders to look up their privileges. Once the privilege is selected, the Message Signing Service signs the email based on the privilege and then with the P-Server's key. The Privilege Admin Service, on the other hand, provides an administrative interface to the privilege administrators to add and revoke users and privileges.

P-Messaging Privilege Verifier

P-Messaging Privilege Verifier (P-Verifier) provides Privilege Verification and email classification services. Upon receiving a privilege signed email, P-Verifier verifies and based on the authorization presented, classifies the email.

The P-Verifier provides two services: (i) Message Authorization Service and (ii) Privilege-list Maintenance. The Message Authorization Service verifies the validity of emails and based on the privileges accepted by a user, classifies the mails into different privilege classes. The Privilege-List Maintenance service provides an interface for user to maintain a list of privileges accepted.

P-Messaging Manager

The P-Messaging Manager is a component that connects to the Privilege Admin service of the P-Server and provides an interface to add and remove

both users and privileges. As discussed above, each P-Server has an administrator who has the capability to maintain the privileges. The administrator has the ability to add and remove privileges. While creating a privilege, the administrator nominates a privilege-owner. The privilege-owner is responsible for maintenance of the users of the privilege. The privilege-owner has the capability to add and delete users from the privilege.

P-Messaging Trust Authority

The P-Messaging Trust Authority is the entity that creates a CoT among P-Servers by providing a certificate to each P-Server. With the help of a digital signature-based mechanism [10], the trust on the senders' P-Server can be verified by the receivers.

Circle of Trust in P-Messaging

P-Messaging provides the capability of verifying a P-Server by peer P-Servers before placing any trust on them with the help of CoT. The CoT among the P-Servers is formed with the help of PKI infrastructure. Honoring a P-Server's privileges, i.e., accepting privileged email to be valid, across domains is dependent upon the maintenance of the trust placed upon the P-Server by the P-Messaging Trust Authority. If a P-Server sends an unsolicited email, the trust placed on it by the Trust Authority can be revoked. To be able to send authorized emails to other P-Servers that are a part of the CoT, a P-Server would strive to be a part of the CoT.

With the help of privilege verification, a recipient can first authorize and then classify the incoming emails into the privilege classes. If an email whose privilege is not subscribed to by the user is received, it is placed into an underprivileged class. However, if a sender's P-Server is not in the CoT, the message is placed into the no-privilege class.

The following subsection describes the various methods in which a CoT is created and maintained in order to support Privilege Messaging.

Addition of a P-Server to the CoT

Figure 2 describes the hierarchical structure of P-Messaging, where the P-Messaging Trust Authority forms an entity that functions as a CA. We make the assumption that the P-Messaging Trust Authority is trusted by all. Each P-Server must receive a certificate from the P-Messaging Trust Authority that is used to verify the P-Server. Each P-Server in turn maintains or moderates multiple privileges. Each of the privileges has its own PKI key pair capable of signing the emails.

Creation of the CoT requires each P-Server to store the privilege's private keys in a secure manner: the key store that holds the private key of the privileges should be maintained securely; i.e., in case of an unauthorized access, the keys must be destroyed. In case of the unauthorized access of the private key of the privilege, the administrator can easily revoke the privilege and create a new PKI key pair for the privilege. We discuss revocation policy adopted a CoT in the next section.

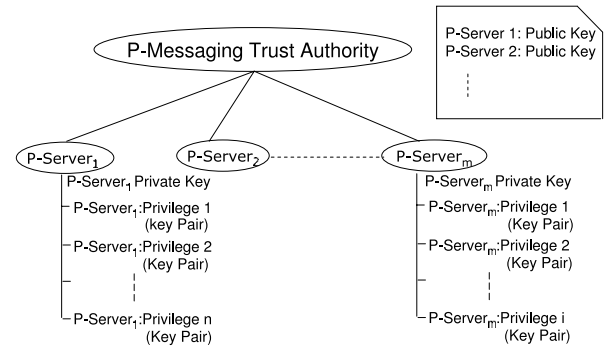


Figure 2: Circle of trust among the Privilege Servers. The P-Messaging Trust Authority allows a Privilege Server to be verified by other Privilege Servers.

Revocation of a P-Server from the CoT

The P-Messaging Trust Authority has the authority to revoke a P-Server based on an issuing agreement of the certificate. One of the attributes of the issuing agreement could be the number of instances an unsolicited mail is reported by users before revoking a P-Server. Once revoked, the P-Server can request a new certificate from the P-Messaging Trust Authority. The P-Messaging Trust Authority can place additional constraints on the P-Server before issuing a new certificate. This paper discusses the mechanism involved to revoke a P-Server; the pre-stipulated policies for revocation are beyond the scope of this paper.

Upon compromise of a privilege, the P-Server administrator is responsible for revoking that privilege. If the privilege is not revoked, the P-Server itself will be revoked from the CoT, thereby invalidating all the privileges held by it. Hence, it is contingent upon the P-Server administrators to maintain the trust placed upon it. A P-Server is responsible for maintaining the legitimate users for each privilege maintained by the P-Server and is delegated to the privilege-owner.

To reiterate, the negative characteristics of a privilege-user will flow upward to their privilege, where if the user is not revoked by the privilege, the privilege is revoked by the P-Server. If the P-Server would not revoke the privilege, the P-Messaging Trust Authority will revoke the P-Server.

Advantages of CoT

With the help of CoT, a P-Server can be verified by peer P-Servers. In other words, as shown in Figure 2, each P-Server acts as a CA for the multiple privileges maintained by it. With the help of CoT, each P-Server independently possesses the capability to create and maintain the privileges that are associated with it. Honoring of a privilege is based on the trust placed on the P-Server by the P-Messaging Trust Authority. This provides distributed authorization among P-Servers where each P-Server is capable of creating its own privileges. To reiterate, with the help of CoT, a scalable architecture with fine-granular email authorization can be provided.

P-Messaging Architecture

As discussed in earlier sections, P-Messaging provides message verification, thereby classifying emails based upon the privileges held. With the help of P-Messaging as shown in Figure 4, legitimate messages can be honored across domains where each domain is managing multiple P-Servers.

Privilege Messaging allows users to send and receive the messages. In the sender architecture, the P-Server attaches privileges on behalf of the user. In the receiver architecture, these privileges are verified before being delivered to the receiver. The following sections discuss the two architectures in further detail.

Sender Architecture

As shown in Figure 3, when sending an email the P-Server first verifies the sender, for instance, Bob. After verification of the user, the P-Server signs the mail with the privileges requested by Bob. The user, Bob, can select a privilege or the P-Server can select a privilege with the help of a simple rule-based engine. The privilege is selected from the Member List maintained for every user at the P-Server. Once the message is signed with the privilege, the message is then signed by the P-Server itself, before relaying it through the MTA to the users who accept the said privilege.

This way, a receiving P-Server can verify the sender P-Server and place trust on the P-Server and then verify the privilege. For example, when a P-Server is installed for a university, the P-Messaging Trust Authority creates a key pair for the P-Server that is securely transmitted. The university P-Server can then create multiple privileges, for example, faculty and student privileges.

For a receiver to accept a message, the receiver should honor the sender's privilege. Without this, the message that is sent cannot be classified into privilege classes but into the underprivileged class. These classes are described in later sections.

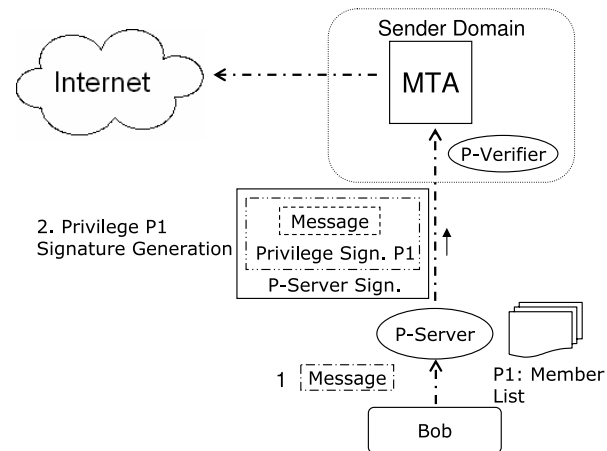


Figure 3: P-Messaging Sender Architecture: the sender Bob is verified, the P-Server signs the message using a privilege specified in the Member List. The mail is then sent from the P-Server to the MTA that relays the email.

Receiver Architecture

Figure 4 shows the receiver architecture in detail. On the receiver's domain, the MTA, upon receiving the email, verifies the privileges with the help of the P-Messaging Privilege Verifier. For verifying a mail, the P-Messaging Privilege Verifier first verifies the P-Server signature, thereby checking the authenticity of the P-Server in the CoT. Once the sender's P-Server is verified, the privilege's public key is retrieved from the P-Server and the email is verified.

Once the mail is verified, the next step is to place the mail into classes. This is performed by checking the user's Privilege List, which contains all the privileges that are accepted by a user. If the receiver honors a privilege, the mail is classified into the privilege class. The email classification is based on the privilege information in the email's header. If the mail is not

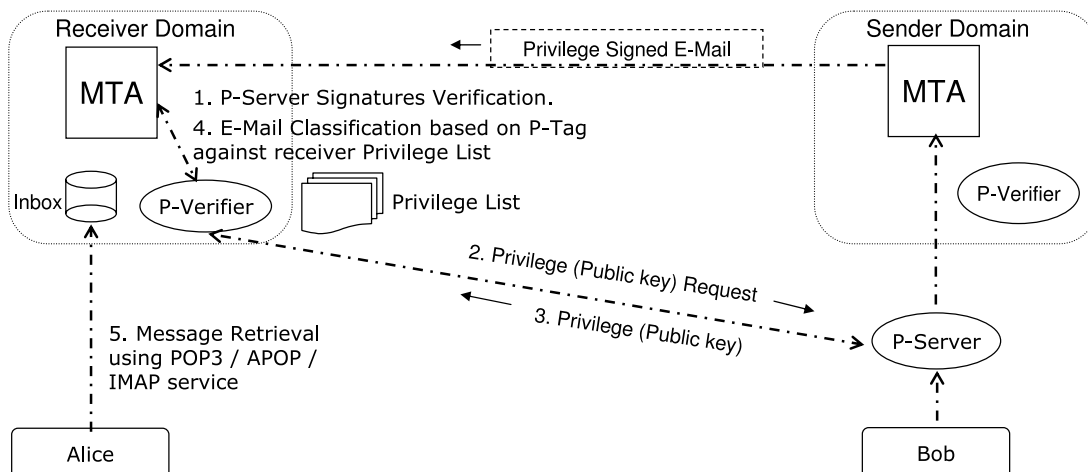


Figure 4: P-Messaging Receiver Architecture: Once an email is received, the MTA passes the mail to the P-Verifier that looks up the public key of the privilege to verify the mail. Once the email is verified, it is classified according to the receiver's Privilege List.

honored by the user, the mail is classified into an underprivileged class. If, however, the message is not verified or signed, the message is placed into the unsigned class. We further discuss the privilege classes in the sections below.

To retrieve the message, as shown in Figure 4, the client, for instance Alice, connects to the mail server to retrieve the messages. Using the additional header information, any email client can display the information in any desired format. The mail clients can show the different ‘inboxes,’ where each inbox caters to a different class. In this way, the classification of the email into different classes provides users with the ability to view the messages according to the privileges accepted by them. This allows a faster lookup for the emails by classifying the emails at one location thereby providing QoS for the users.

Recursive Privilege

Another benefit of P-Messaging would be to request a privilege from another P-Server on the basis of a privilege that is held by the email. For example, as shown in Figure 5, a user in the UNCC domain requires a USENIX privilege with an email. The UNCC privilege would not be accepted by, suppose, LISA committee. However, on the basis of Bob’s Privilege, the LISA privilege can be obtained. The LISA privilege can be used to communicate with other users of the Privilege -LISA. Figure 5 shows the Sender architecture for Recursive Privilege mechanism. The advantage of such a technique is that users can sign using their privileges across another administrative domain before sending an email. Another example would be a user using a free email ID to sign the mail using the class privilege to send the mail to the faculty who teaches the course at a university. A single mail can thus have multiple privileges, demonstrating to the receiver the sender’s multiple credentials.

Recursive Privilege requires a user to be a member of a privilege across domains. The member list contains the list of members who are authorized to send an email using the privilege. A privilege can be created for each user for enabling cross domain privileges. This enables users to attach a privilege as a single user rather than the complete group. While requesting an email, the P-Server sends the Privilege-tag information of the first privilege. Instead of transmitting the complete message to the secondary P-Server, only the privilege information can be sent. This requires that only a small amount of information be transmitted over the wire to receive additional privileges. After verification of the privilege, the peer P-Server verifies the privileges and attaches its own privilege. Moreover, the verification of the privileges will be based on recursive verification of each privilege, allowing users to trace the order of privilege selection.

P-Messaging Classes

P-Messaging defines multiple Privilege Classes for the emails classified for the receivers. These classes take into consideration the credentials presented by the email as well as the receivers preferences. The user preferences indicate the list of the privileges that are further honored after the email has been verified. As described earlier, P-Messaging provides QoS with the help of this classification. We define the Privilege Classes into three categories.

Privileged Classes

Privileged classes contain the emails that are successfully verified and honored by the receiver, are placed. The emails can be further classified based on the privileges that it is associated with it.

Underprivileged Class

Underprivileged classes contain the emails that are verified, but the associated privileges are not honored

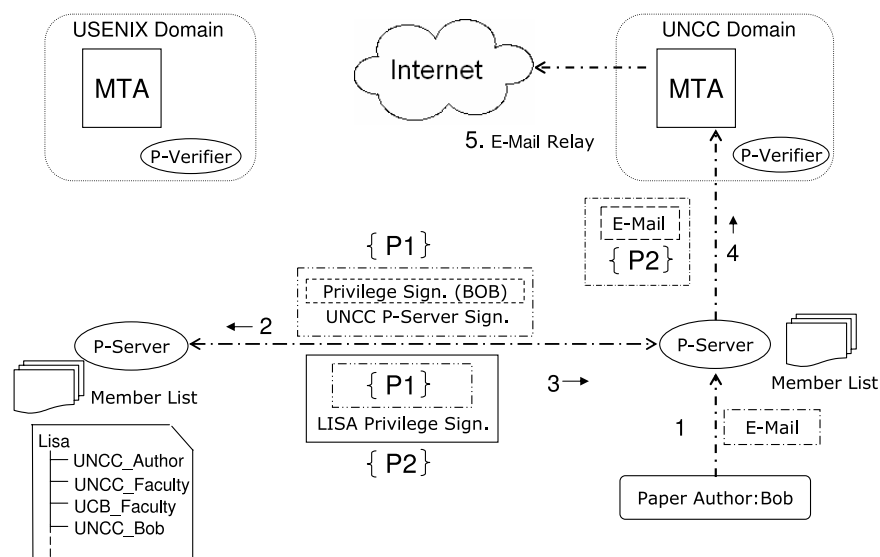


Figure 5: The Recursive Privilege Architecture showing multiple Privileges being attached to an email.

by the receiver. If a privilege presented by an email is deemed important, the receiver may subscribe to the privilege. An email from a sender with a privilege that is not honored will be placed in this class. Once a user honors a privilege, it will be placed into the Privileged Class.

No-privilege Class

The no-privilege classes form the lowest class among the privilege classes, where unsigned or emails whose authenticity cannot be ascertained are placed. As P-Messaging becomes widely accepted, the number of mails in the no-privilege class would be reduced.

Privilege Tag

Each email possesses credentials that allow the email to be classified into different classes. These credentials, referred to as Privilege Tag (P-Tag), provide the users with the information about the sender with the help of a digital signature. With the help of digital signature, Privilege Messaging demonstrates the authenticity of the email's origin.

In this section, we describe the format of the P-Tag and the various interfaces that are required to maintain the privilege.

Extensible P-Tag

As described above, the P-Tag information contains the privilege's digital signature. P-Messaging Tag management is extensible, so that each P-Server creates its own privileges. Each P-Server maintains privileges' public keys for other P-Servers to verify a privilege. Thus, each P-Server acts as a CA for the privileges it holds. The P-Tag format is as follows:

```
[P-Server]:[Privilege]
```

The P-Tag information is appended as a part of the email header. Conceptually, the following is the structure of a Privileged message:

```
{[Tagged Email] Privilege Signature}:
{[Privilege Signature]
  P-Server signature}
```

The privilege signature is created on the email. The P-Server signs the Privilege Signature. Hence, to verify a privileged email, first the P-Server signature is verified. Once the P-Server signature is verified, the Privilege Signature needs to be verified. In the case of Recursive Privilege, the P-Tag information is shown below:

```
{[Tagged Email] Privilege Signature}:
{P-Tag 1}: ({P-Tag 1 }P-Tag 2) ...
Where P-Tag n is {[Privilege Signature]
  P-Server signature}
```

As discussed in sections above, in the recursive Privilege-tag assignment, multiple privileges are attached, based on the privileges already presented by it. The complete message need not be transmitted to the second P-Server, as only the P-tag information is needed to verify the sender of the privilege to identify and attach a new privilege to it. The next few sections describe the method for creating and maintaining the P-Tag information.

P-Tag Creation and Maintenance

As part of Privilege Management, apart from creation and maintenance of the privileges, a privilege-owner performs the tasks of adding and deleting/revocation of users. The privilege-owner is also responsible for:

- Addition of Privilege to user.
- Deletion or Revocation of a user's Privilege.

Addition and revocation of privileges deal with modifying the Member List for a user. The Member List, as discussed below, contains the privileges a user is authorized to send with. A user can be added or revoked to the Member List only by the privilege-owner. If a user wishes to be added or deleted, a request should be sent to the privilege-owner. If the privilege-owner considers the request, the Member List will be updated at the P-Server; otherwise, the request is rejected.

However, if a user abuses the privilege, the privilege-owner should revoke the user. The revocation of the user will be performed by removing the user from the Member List. As the private key of the privilege is not revealed to the user, the privilege-owner need not create another PKI key pair for the privilege.

Privilege-List Maintenance

Each user maintains the Privilege List at the P-Verifier. This information needs to be updated by the user to classify the emails based on the privileges listed in the Privilege List. If a user wishes to honor a privilege, the user updates the privilege list with the P-Verifier. Adding a privilege to the privilege list is similar to maintaining white-lists albeit at the server side.

To assist users with initial list of privileges, a default list of privileges can be assigned to users by the

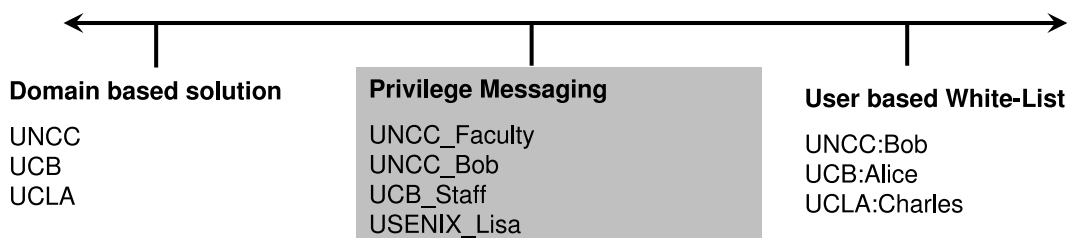


Figure 6: Example credentials maintained in different technologies.

mail service provider. This allows a default privileges associated with a user during the setup phase. In the absence of a user's input, which we believe quite common, the service provider's default list will be used. Some user-profiling and personalization techniques may be useful in determining the list on behalf of the user.

P-Messaging: Design and Benefits

This section discusses the various design decisions for Privilege Messaging. Using the CoT, Privilege Messaging provides a QoS over the current email architecture. The additional benefits are that P-Messaging would allow fine-granular privilege maintenance which allows the negative reputation to be contained within a privilege rather than the domain. Having the privilege in the white-list as compared to individual email IDs allows smaller list to be maintained. Also, a new correspondent will not be considered unsolicited sender.

Fine-Granular Reputation Management Than the Domain-Based Solutions

As discussed in the related work, P-Messaging is a solution falling between domain-based solutions and the user based white-lists in terms of its granularity. Figure 6 shows examples of the credentials for the different technologies. In the case of the domain-based solutions, the unit of credential is a domain. For P-Messaging, the credential is maintained per P-Tag, which can contain either single user or a group of users.

A domain-based solution publishes the credentials in the DNS records; therefore each mail sent has a single credential for the entire domain. With domain-based solutions, when a domain is registered it will be given a full authorization to send a message to other domains. However, when P-Server is registered, the domain will be given an authorization to issue and manage the P-Tag, not the right (authorization) to send the message. Privilege Messaging can be installed over multiple P-Servers on a domain where each P-Server maintains multiple privileges. Having multiple privileges allows negative reputations to be contained to a single privilege rather than a complete privilege server.

DKIM allows public keys to be created and embedded into the DNS records by the mail server itself, whereas Privilege Messaging requires the P-Server to publish its public key with a CA, the P-Messaging Trust Authority. Due to CoT, the trust on a sender is verified before the message is accepted at the receiver. In other domain-based solutions, the message is accepted without verification by a trusted third party.

In P-Messaging, the negative reputation is constrained into a single privilege as compared to the complete domain. Furthermore, P-Messaging provides QoS by automatically classifying the emails. This is further discussed in the next subsection.

Privilege Messaging has multiple keys that are required to classify emails. As discussed above, P-Server

and its privileges are associated with their own corresponding PKI key pair. In case of the loss of the privilege key, the privilege-owner can request the administrator to reissue a new PKI pair for the privilege. If the P-Server's key is lost, the administrator requests the P-Messaging Trust Authority with a new PKI key pair.

Maintainable White-list Using Privilege-ID Rather Than Individual Email ID

The credentials for the user-based white-lists, as shown in Figure 6, are the individual email IDs. As discussed earlier, the privileges are the credentials in Privilege Messaging. In comparison to user-based white lists, the mails are not classified based on the sender's privilege or an email contents in Privilege Messaging.

The deployment of P-Messaging is valuable to an organization since the user key maintenance [16] is eliminated by maintaining the keys on an infrastructure level. It needs to be noted that PGP can still be used in conjunction with P-Messaging on a user-level basis for providing confidential and integrity services for an individual.

With white-lists, a new correspondent might be classified as an unsolicited sender. The benefit of P-Messaging is that a new correspondent may not be classified as an unsolicited sender.

Automatic Email Classification Based on the P-Tag

Based on the privileges presented, the emails are classified automatically based on the privileges that are accepted by the users. The automatic classification of emails, depending on the sender privileges and those that are honored by the receiver, provides the ability to classify the mails into three classes: the privileged class, underprivileged class and no-privilege class. The classification provides the QoS by allowing similar emails to be checked in one location.

In addition to such automatic classification, email clients can create multiple 'inboxes' based on accepted P-Tags. For example, the end-user email client can move a message signed by USENIX privilege into USENIX folder based on a user-specified rule. Email clients can be programmed to create separate inboxes for each privilege subscribed. This allows the server-side classification and user-side display of the classified emails.

Retaining the Benefits of Existing Email Architecture

In this section, we discuss the benefits of the current email architecture that are also supported by Privilege Messaging. Privilege Messaging provides sender privacy, incremental deployment over the current email infrastructure and use for the large-scale email service providers.

Sender Privacy

As an additional benefit, P-Messaging provides sender privacy. Once the user is authenticated at the sending P-Server, the sender information can be removed from the email, without the loss of the

privileges associated with the email. Similarly, relaying emails from domains is possible. The privilege-signed email without the header can then be verified and classified based on the privileges. With this ability, users can send email to a person in the group without their information. For instance, the sender information from a student's email in a class can be removed, and the email be delivered to the professor verifying that the email is coming from the class, but not from a specific student. Though the message is sent without the sender information, the message will not be considered as unsolicited using Privilege Messaging.

Certain websites, such as job search service websites, send a notification email to the registered clients. However, due to web-site policy, the sender email ID is changed to the receiver's email ID. Such valid source privacy can be provided by P-Messaging. Sender information can be removed from the emails, yet classification can be easily performed without them being marked as unsolicited.

Incremental Deployment

Privilege Messaging can be deployed incrementally over the current email infrastructure. Through P-Messaging, the unsigned messages will be classified into the no-privilege class. To classify emails from the users into other privilege classes, P-Messaging should be deployed by the sender and the receiver. P-Messaging complements the existing email infrastructure, retaining all the benefits (e.g., relaying) at the same time adding the much-needed authorization framework.

With the large scale adoption of Privilege Messaging, the unsolicited mails sent over the wire will be reduced, allowing the mail to be sent only from verifiable servers. And while much of the transmitted bulk emails would still be mostly unwanted, the source of the sender can be verified and the ability to restrict those emails is also provided to the user.

Large-Scale Email Service Providers

Privilege Messaging allows system administrators to deploy P-Messaging for large scale email providers that have users who abuse the system. The providers can classify their user base into multiple privileges; for example, on the number of years the email ID has been in use and/or the location of the user. This creates smaller subsets of users to be dealt with. If a user abuses their privileges, the email providers can revoke the user and their privileges. Alternatively, the users who wish to use their email accounts with P-Messaging can be provided with 'Signing Services,' which sign the users' email on behalf of the user. Meanwhile, the P-Server that sends the mails can request additional terms with P-Messaging Trust Authority, so that a few negative instances would not revoke the P-Server from the CoT.

P-Messaging Prototype Implementation and Configuration

In this section, we present the implementation of P-Messaging. We discuss the sender and the receiver

configuration, and demonstrate the processes involved in setting up P-Messaging and the process for sending and receiving emails.

Implementation Details

P-Messaging has been implemented using Java 1.4. The P-Messaging Trust Authority uses Remote Method Invocation (RMI) to generate the PKI key pair for the privileges as well as for the P-Server. For transmitting the mail from the client to the P-Server, the RMI architecture is used.

The P-Server uses Java Mail API to transmit the mail to the MTA. The MTA is Sendmail. Sendmail has been chosen as it allows an external entity, a Milter, to classify the messages. A java implementation of Milter is provided using Jilter API [19]. The systems running the mail servers also provide IMAP services using the Cyrus IMAP server. This module essentially functions as the P-Verifier that provides digital signature verification.

Sender and Receiver Configurations

In this section, we discuss the changes to the sender and the receiver side for deploying P-Messaging at an organization. Privilege Messaging allows two different changes to the configurations to the sender and requires a single configuration to the receiver side. In order to use P-Messaging with legacy email servers, the email clients need an added mechanism that shows the listing of privileges at the time of sending. The retrieved emails need to be classified based on the associated P-Tag.

Sender Configuration

As described above, the sender side configuration can be created by two different methods. The first method requires changes to the email client so that the user privileges can be retrieved from the P-Server. With the change in the email client, the user can select the privilege that the email needs to be sent with. The second configuration requires the privilege selection with the help of a simple rule based privilege selection engine where a privilege is selected while sending an email to a specific user/ group of users.

Receiver Configuration

As discussed above, the receiver side configuration is minimal. The receiver side installation is a one time installation of P-Verifier. The P-Verifier, which is a Jilter, needs to be added to the Sendmail configuration file. The following is the configuration lines:

```
O InputMailFilters = <Jilter Name>
X<Jilter Name>, S=inet:<port>@<IP address>
```

The configuration needs to be added to the /etc/mail/sendmail.cf (Fedora Core) configuration file. Figure 7 shows the configuration for Sendmail.

Test Setup

All of the experiments have been performed using the following devices and networks with the specified configurations. The client and the P-Messaging Trust Authority runs on Intel Pentium 4 CPU 3.20

GHz with 1.5 GB RAM running Microsoft Windows XP Professional version 2002.

The two mail servers run Sendmail 8.12.10. The first system is: Intel Pentium 4 CPU 2.5 GHz with 512 MB RAM running Linux 2.6.14 Kernel: this system accepts mails. The second system is an Intel Celeron 2.5 GHz with 1 GB RAM running Linux 2.6.12.6. This system serves as the primary P-Server, the sender domain. The Local Area Network bandwidth was about 100 Mbps with a delay of about 0.1-0.2 milliseconds.

Adding and Revoking a P-Server to the CoT

As discussed in the above sections, maintenance of the CoT is important for a P-Server to place trust on any other entity. Figure 8 shows the process of adding a P-Server to the CoT. The process of adding the P-Server to the CoT involves creation of a PKI key pair. When installing a P-Server, the P-Server asks the necessary questions to create a PKI key pair. Once the information is gathered, this information is sent to the P-Messaging Trust Authority over RMI which creates the key pair.

Another important aspect for maintaining the CoT is the revocation of a P-Server. The process of revocation is carried out at the P-Messaging Trust Authority. The revocation is performed by removing the P-Server

from the trusted list. Figure 9 shows the revocation process of the P-Server. The present prototype implementation of Privilege Messaging does not cache the public keys of the peer P-Server, the verification is done by looking up the P-Messaging Trust Authority for the privilege's public key. Thus, the present version of P-Messaging does not use Certificate Revocation List (CRL) to remove the defaulting P-Server.

Maintenance of the Privilege

As discussed above, each privilege is created by the P-Server administrator and is managed by a privilege-owner. The privilege-owner is capable of creating and deleting a privilege. Once the privilege modifier is started, a user can create a privilege and assign a privilege-owner. Privilege creation involves the creation of a PKI key pair with the predefined site information. Once a privilege is created, users can be added to the privilege's member list. Figure 10 shows the privilege management.

Privilege List Maintenance

Apart from maintaining the Member-list, a user needs to maintain the Privilege List. The Privilege List is a list maintained at the P-Verifier by each user. The list contains all the privileges that are accepted by the

```
# #####  
#####  
#####  
#####  
#####      MAIL FILTER DEFINITIONS  
#####  
#####  
#####  
#####  
  
XpMess, S=inet:999@152.15.97.77  
#^L
```

Figure 7: Jilter Configuration (P-Verifier) at the receiver domain.

```
C:\Messaging\classes>java pms.PrivilegeServer
Privilege Server Not registered.
To register enter following details:
Please enter a UserName for the Privilege Server:
admin
Organization Unit      :      isr
Organization Name      :      uncc
City                   :      charlotte
State                  :      nc
Country                :      usa
Privilege Server registered with the alias: ISR04
Privilege Server started working
```

Figure 8: Registration of Privilege Server with P-Messaging Trust Authority which generates Public key for Privilege Server.

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\PMessaging\classes>java PMTA.PSRevoke

Enter Alias of the Privilege Server to be removed: ISR04
Privilege Server with alias ISR04 and its corresponding public key is removed fr
om the trusted list.

C:\PMessaging\classes>
```

Figure 9: Revocation of Privilege Server at P-Messaging Trust Authority. The Public key cannot be retrieved by peer Privilege Servers after revocation.

user. Figure 11 shows the interface where a user can honor or dishonor a privilege from the privilege list. The privilege list maintenance is implemented using RMI.

Apart from privilege creation, the P-Server administrator should be able to revoke a privilege from the list. Revocation of a privilege involves removing the users from the privilege and modifying the Member-list. The PKI key pair is invalidated so that the users can no longer use the privilege.

Sending a Privileged Email

Figure 12 shows the method in which a mail is sent by a user using our initial prototype. Once the user selects 'Send Email' from the client interface, the interface shows all the privileges that can be used to send an email with. Once the privilege is selected, the message is then sent to the P-Server. The P-Server adds the privilege signature to the email and sends the email out.

Classification of the Email By P-Verifier

Figure 13 shows email classification by P-Verifier. As described in previous sections, P-Verifier is a Jilter interface that verifies the emails based on the privilege information provided by the email.

P-Verifier classifies the mail, as shown in the Figure 13; the Jilter accepts the emails on a specified IP address and the port. The Output is shown in the following format:

```
<Receiver> <Sender> <Subject>:
<Privilege Verification Status>
```

Retrieval of Email By an Email Client

Figure 14 shows the mechanism used to retrieve the mail. The email client is a prototype model that demonstrates the classification of emails that are classified by the P-Verifier. Once the emails are retrieved, the messages are classified based on the attached P-Tag.

An email client that creates multiple 'inboxes' for each privilege for a user would enable access to the emails in one location. Such a service would provide QoS for the users who would like to access all their emails at one location.

Performance Results

This section demonstrates the performance of P-Messaging. The results were performed on the same configurations as described earlier.

```
C:\PMessaging\classes>java pms.PSAdmin
Welcome to Privilege Server Administration

Select one of the following options:
1. Create a User.
2. Create a Privilege.
3. Delete a Privilege.
4. Exit.
>2
Enter the name of the Privileges to be created: class6102
Enter the Privilege owner ID: gsingara@coiti598.uncc.edu

Privilege with name class6102 created.
=====
```

Figure 10: Privilege Server Manager Interface that connects to PS Admin service to create and delete a privilege.

```
C:\PMessaging\classes>java pmsClient.PrivilegeClient
Please input the UserName...
>bbkang@coiti300.uncc.edu
Please input password...
*****
Select one of the following options:
1: Send Email
2: Get Email
3: Honor Privilege
4: Dishonor Privilege
5: Quit
>3
Enter the name of the privilege to be honored : class6102
Enter the name of privilege handling domain: coiti598.uncc.edu
Privilege [coiti598.uncc.edu:class6102] added to your Privilege List
-----Privilege List Updated-----
Select one of the following options:
1: Send Email
2: Get Email
3: Honor Privilege
4: Dishonor Privilege
5: Quit
>4
Enter the name of the privilege to be dishonored : class1141
Enter the name of privilege handling domain: coiti598.uncc.edu
Privilege [coiti598.uncc.edu:class1141] removed from your Privilege List
-----Privilege List Updated-----
```

Figure 11: Client Interface that connects to Privilege Server to honor or dishonor a privilege for a user. To honor or dishonor a privilege at client, the information about the Privilege Server that handles the privilege is required. The example above shows a professor managing two different privileges: class1141 and class6102.

Our experiments were geared towards demonstrating the performance costs associated with Privilege Messaging compared with PGP-signed email. To determine the P-Messaging performance, we conducted the following experiments:

1. P-Tag Generation Time
2. P-Tag Verification Time
3. Privilege Generation and Verification Time

P-Tag Generation Results

To demonstrate the overhead incurred due to generation of P-Tags, we compared P-Messaging's tag generation performance with the time taken to generate PGP digital signature and unsigned emails. Figure 15 shows our results. The time taken to generate the tag was reasonably higher PGP and unsigned messages, the overhead grows linearly. The overhead includes the time taken to request for a privilege using RMI and generation of two signatures by the P-Server. The result that is shown in the Figure 15 is expected as the P-Messaging performs a double signature and takes twice the time as compared to the time taken to generate PGP signature.

P-Tag Verification Results

To demonstrate the overhead incurred due to verification of the privilege, we compared the time taken

to verify the Privilege with time taken to verify a mail using PGP. Our results are shown in Figure 16 where the time taken to verify the emails is twice the time taken to verify the PGP signed mail. The results are as expected since the Privilege signed mails include two signatures as compared to one signature in PGP.

Privilege Generation and Verification Time

Our experiments show that the time taken to generate a Privilege-tag for an email and send it over LAN was about 0.16 sec. This time included the time taken to generate double signatures: one for the privilege and another for the P-Server server. The time taken to verify a message was about 0.09 Sec, again this time involved the time taken to verifying the P-Server signature, and then the Privilege signature. It also involved the time to retrieve the privileges' public key from the sender's P-Server.

Future Work

This section discusses the future work for Privilege Messaging. For P-Messaging to be effective, we further need a mechanism that would allow the receiver to evaluate the Privilege-tags along with the corresponding P-Server. In order to prevent a privilege from being used for sending unsolicited content, a

```
C:\PMessaging\classes>java pmsClient.PrivilegeClient
Please input the UserName...
>sumeet@coiti598.uncc.edu
Please input password...
*****
Select one of the following options:
1: Send Email
2: Get Email
3: Honor Privilege
4: Dishonor Privilege
5: Quit
>1
=====
-----Sending Message-----
=====
Select one from the following available Privileges:
1: SIS_Dept
2: student
3: class6445
4: class6102
>class6120
To(when done, enter '.' on a new line):
bbkang@coiti300.uncc.edu
.
Subject: Sample Mail
Message: Sample mail with privilege
=====
-----Privilege Mail Sent-----
=====
```

Figure 12: Client interface to connect to a Privilege Server to send an email. The client required to select a privilege to send an email.

```
[root@coiti300 classes]# java com.sendmail.jilter.samples.standalone.SimpleJilterServer -c pMess -p ine
t:999@152.15.97.77
[To] [From] [Subject] [Classification]
[gsingara@coiti300.uncc.edu] [sumeet@coiti598.uncc.edu] [document] [Privilege Verified]
[pratik@coiti300.uncc.edu] [sumeet@coiti598.uncc.edu] [discription] [No-Privilege]
[bbkang@coiti300.uncc.edu] [sumeet@coiti598.uncc.edu] [Sample Mail] [Privilege Verified]
[bbkang@coiti300.uncc.edu] [gsingara@coiti598.uncc.edu] [image names] [Under Privilege]
[pratik@coiti300.uncc.edu] [sjain9@uncc.edu] [test mail2] [No-Privilege]
[bbkang@coiti300.uncc.edu] [sumeet@coiti598.uncc.edu] [new name] [No-Privilege]
[bbkang@coiti300.uncc.edu] [gsingara@coiti598.uncc.edu] [Deployment issues] [Privilege Verified]
[pratik@coiti300.uncc.edu] [jain.sumeet@yahoo.com] [sample mail] [No-Privilege]
```

Figure 13: Demonstration of P-Verifier interface in a verbose mode for the classification of emails at the receiver domain.

reputation system needs to be developed so that the reputation values of P-servers and their privileges are periodically updated with an increase for right behavior and a decrease for negative behavior. Such a reputation system in a distributed environment with partially trusted entities is difficult to achieve, since each partially-trusted server might hold varying number of privileges each with varying number of users. The negative reputation of a privilege can propagate to the server and therefore the reputation of all privileges associated might be affected. Hence, an effective reputation management would be essential for the successful adoption of P-Messaging. The reputation value should be embedded into a certificate for the P-Server and the Privilege-tag.

Privilege Management requires better interfaces for Privilege-tag management. A usage-study on the

ease of P-Messaging usage would allow a better UI design or additional mechanisms for Privilege Management. For instance, consider a case when the P-Tag owner receives a request for addition to the privilege. The P-Tag owner will give access only to those users who can successfully provide their identity: the requestors should themselves be a part of some verifiable Privilege Server, thereby reducing the number of requests received by the owner.

Privilege selection is an important aspect to be performed by the senders. For a message to be created in a receiver's mailbox, P-Messaging mandates that the sender should have at least one privilege that the receiver would accept. A protocol should exist that pre-computes a privilege that a sender would require so that a message can be accepted at the receiver. This protocol can perform a data mining on the privileges

```
C:\PMessaging\classes>java pmsClient.PrivilegeClient

Please input the UserName...
>bbkang@coiti3000.uncc.edu
Please input password...
*****
Select one of the following options:
1: Send Email
2: Get Email
3: Honor Privilege
4: Dishonor Privilege
5: Quit
>2

Privileged Mails
-----
[Number] [Subject] [From]
[1] [test mail] [pratik@coiti598.uncc.edu]
[2] [Sample Mail] [smeet@coiti598.uncc.edu]
[3] [Deployment issues] [gsingara@coiti598.uncc.edu]

Underprivileged Mails
-----
[Number] [Subject] [From]
[4] [image names] [gsingara@coiti598.uncc.edu]

No-privilege Mails
-----
[Number] [Subject] [From]
[5] [new name] [smeet@coiti598.uncc.edu]

Enter the Message number you want to read: 2
Mail Content:-
Sample mail with privilege
=====
-----End of Mail-----
=====
```

Figure 14: Prototype client implementation allowing user to retrieve classified emails.

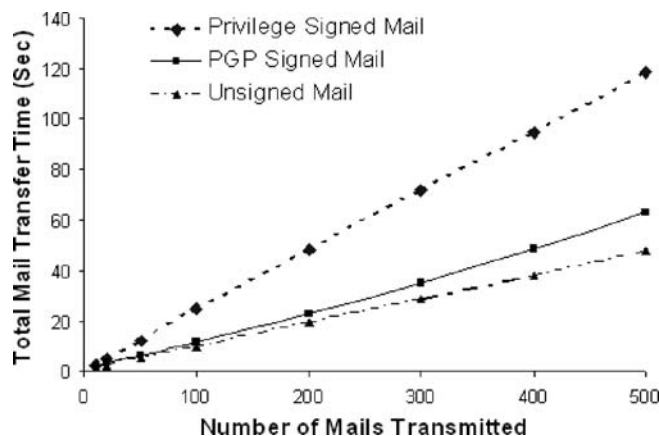


Figure 15: Comparing time taken for sending P-Tag attached emails, emails with PGP signature and unsigned emails.

of the sender and the receiver to select a common privilege. Data mining can be useful for recursive-privilege mechanism where the senders can add additional credentials, which they hold on peer P-Server. The additional credentials are created based on the privileges that the receiver honors.

The present architecture does not cache public keys for a P-Server and its privileges. Caching the public keys allows faster verification of emails. Introducing key caching requires Certificate Revocation List (CRL) management. CRL management would be challenging in a distributed architecture for revoking the certificate of a privilege maintained by peer P-Server. A better privilege verification would consider reducing the number of round trips required to verify the privilege. Also, previously sent emails need to be verified in the event where the Privilege's key has been revoked.

Conclusion

In this paper, we presented an authorization framework, called Privilege Messaging, overlaying the existing email infrastructure while retaining the beneficial aspects such as relaying. For the sender, P-Messaging provides a mechanism that allows email delivery only if the sender possesses the privilege that the receiver would accept. Based on the email privileges, the email is classified automatically according to the Privilege Tag at the receiver, providing QoS for the user. Having the privileges in the white-list as compared to individual email IDs allows a smaller list to be maintained and a new correspondent may not be regarded as an unsolicited.

Each Privilege Server manages multiple privileges as opposed to a single credential as previously proposed in domain-based authentication schemes. In case of the compromise or spam being propagated from one domain, the negative reputation is contained within a privilege rather than the complete domain.

With the help of P-Messaging, the email's authenticity is verified by a trusted third party. To allow

privileges to be verified across domains, P-Messaging establishes a Circle of Trust among the P-Server for privilege verification. P-Messaging performs dual digital signature on an email, first by the assigned privilege and then by P-Server, allowing peers in the CoT to verify the email's authenticity. This ensures that only authorized users can send messages only if their P-Server is a member of CoT, and that a P-Server needs to limit the unwanted email that it transmits or it would be revoked from the CoT.

Privilege Messaging can be deployed incrementally; P-Messaging is a gradual process of introducing the authorization over the current email infrastructure. To support such deployment, P-Messaging can coexist with other technologies, providing trust-based email service over MTA. P-Messaging is designed to work well over the existing SMTP infrastructure with minimal user-level interaction and deployment overhead for the authorization provided.

Finally, for more information, please visit: <http://isr.uncc.edu/pmessaging>.

Acknowledgment

We would like to thank our shepherd, Rowan Littell, and the anonymous reviewers for their insightful comments. We would also like to show our appreciation to our LISA copy-editor, Rob Kolstad, for his excellent service. Finally, we thank our ISR lab members for their help from the early stage of this paper.

Author Biographies

Brent Hoon Kang received his Ph.D in Computer Science from the University of California at Berkeley, working on the Berkeley Digital Library and OceanStore project. Prior to Berkeley, he received an M.S in Computer Science from the University of Maryland at College Park, and a B.S in Computer Science and Statistics from Seoul National University with 1st place distinction among computer science majors. Since Fall 2004, he has been an assistant professor at the University of North Carolina (UNC) at

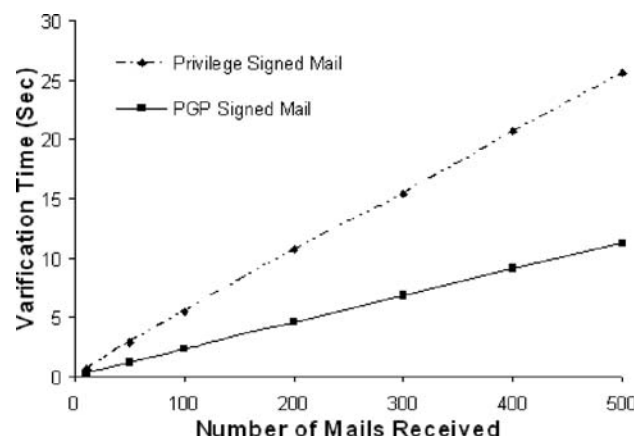


Figure 16: Comparing time taken for verifying emails with P-Tags and PGP signed emails.

Charlotte. He is currently leading the Infrastructure Systems Research (ISR) Lab with a focus on securely architecting large-scale infrastructure systems, working with five graduate students. Hoon can be reached at bbkang@unc.edu.

Gautam Singaraju is a fourth year doctoral student advised by Dr. Kang. He completed an M.S in Computer Science at UNC Charlotte in 2003. Since then, he has also been a volunteer System Administrator for a global non-profit organization. Gautam can be reached at gsgingara@unc.edu.

Sumeet Jain is a second year graduate student working on his master degree with Dr. Kang at UNC Charlotte. He completed an M.S. in Computer Science at Rajiv Gandhi Technical University, India, and worked with Choksi Laboratories Limited as a Software Developer. Sumeet can be reached at sjain9@unc.edu.

Bibliography

- [1] Ahmed, S., F. Mithun, "Word stemming to enhance spam filtering," *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004.
- [2] Allman, E., *DomainKeys Identified Mail (DKIM): Introduction and Overview*, 2005, <http://mipassoc.org/dkim/info/DKIM-Intro-Allman.html>.
- [3] Andreolini, M., M. Colajanni, F. Mazzoni, L. Messori, "HoneySpam: Honeypots fighting spam at the source," *Proc. USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop*, Cambridge, 2005.
- [4] *CAN-SPAM Act: Requirements for Commercial Emailers*, <http://www.ftc.gov/bcp/online/pubs/buspubs/canspam.htm>.
- [5] Duan, Z., K. Gopalan, Y. Dong, "Push vs. Pull: Implications of Protocol Design on Controlling Unwanted Traffic," *Proc. USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop*, Cambridge, 2005.
- [6] Finnegan, O., *Email Deliverability Getting your Email into the inbox*, 2005, http://www.ieinternet.com/mailwall/Email_Deliverability_whitepaper.pdf.
- [7] Gomes, L. H., C. Cazita, J. M. Almeida, V. Almeida, W. Meira, "Characterizing spam traffic," *Proc. 4th ACM SIGCOMM Conference on Internet Measurement*, 2004.
- [8] Gray, A., M. Haahr, "Personalised, Collaborative spam filtering," *Proceedings the First Conference on Email and Anti-Spam (CEAS)*, 2004.
- [9] Hardy, I. R., *The Evolution of ARPANET Email*, Thesis, Department of History, University of California, 1996.
- [10] Kolcz, A., A. Chowdhury, J. Alspector, "The impact of feature selection on signature-driven spam detection," *Proc. the First Conference on Email and Anti-Spam (CEAS)*, 2004.
- [11] Leiba, B., N. Borenstein, "A multifaceted approach to spam reduction," *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004.
- [12] Microsoft Corporation, *Sender ID Framework – Executive Overview*, 2004.
- [13] Milletary, J., *Technical Trends in Phishing Attacks*, 2006, http://www.cert.org/archive/pdf/Phishing_trends.pdf.
- [14] NACHA, "Phishing losses total \$500 million," Technical report, NACHA – The Electronic Payments Association, 2004.
- [15] Neustaedter, C., A. J. Bernheim Brush, Marc A. Smith, Danyel Fisher, "The Social Network and Relationship Finder: Social Sorting for Email Triage," *Proc. Conference on Email and Anti-Spam (CEAS)*, 2005.
- [16] Price, W., *Inside PGP Key Reconstruction*, A PGP corporation White paper, 2003.
- [17] Realtime Blackhole List, Mail Abuse Prevention System LLC, California, 2002, <http://www.mail-abuse.org/rbl/>.
- [18] Segal, R., J. Crawford, J. Kephart, B. Leiba, "Spamguru: An enterprise anti-spam filtering system," *Proc. of the First Conference on Email and Anti-Spam (CEAS)*, 2004.
- [19] *Sendmail-Jilter API*, 2005, <http://sendmail-jilter.sourceforge.net/index.html>.
- [20] W. Wong, M., *Sender Authentication: What to do*, Technical Document, 2004, <http://www.openspf.org/whitepaper.pdf>.
- [21] Yahoo Inc., *DomainKeys: Proving and Protecting Email Sender Identity*, <http://antispam.yahoo.com/domainkeys>.
- [22] Zimmermann, P., *The Official PGP User's Guide*, MIT Press, Cambridge, 1995.

Securing Electronic Mail on the National Research and Academic Network of Italy

Roberto Cecchini – INFN, Florence; *Fulvia Costa* – INFN, Padua;
Alberto D'Ambrosio – INFN, Turin; *Domenico Diacono* – INFN, Bari;
Giacomo Fazio – INAF, Palermo; *Antonio Forte* – INFN, Rome;
Matteo Genghini – IASF, Bologna; *Michele Michelotto* – INFN, Padua;
Ombretta Pinazza – INFN, Bologna; *Alfonso Sparano* – University of Salerno

ABSTRACT

Sec-mail is a group of site administrators in the GARR (Gruppo Armonizzazione Reti della Ricerca – Research Networks Harmonisation Group) network dedicated to the security of E-mail services. GARR is the National Research and Academic Network of Italy.

The main points covered are in: methodologies to improve the efficiency of spam detection (mainly tuning of SpamAssassin), definition of best practices in electronic mail administration, sender domain authentication, greylisting and spam monitoring.

Introduction

The National Research and Academic Network of Italy, or GARR, formed the working group SEC-MAIL to study IT security related problems on its network. This group was formed following a proposal by Roberto Cecchini at the V GARR Workshop held in Rome in November 2003. The group examines the following technologies:

- spam;
- viruses and worms spread up by E-Mail;
- best practices for mail-server configuration and security;
- authentication of sender mail-server;
- greylisting technologies;
- graphs and statistics.

This paper presents a summary of the group's findings to date. Since the fall of 2004, the group has its own web area and a wiki site [WSM], where both the results of the experiments and the produced documentation are made available.

Initially, the group focused its work mainly in SpamAssassin tuning in order to improve the spam detection by Bayesian filters, non-standard rules, and technologies based on mass-emailing distributed identification. Some experimental DCC (Distributed Checksum Clearinghouse) servers were set-up, and made available to the GARR community on a best-effort basis. Then, greylisting has been enabled on three pilot sites, and with an experimental technology in one of them. A huge amount of graphs and statistics has been produced which have surely helped the several experimental activities of our working group.

We will discuss our efforts to prevent spam and then discuss viruses and worms. This is followed by best practices, Sender domain authentication, greylisting technologies, and a conclusion.

Controlling Spam

Spam: A Security Problem

In the past years the number of unsolicited E-Mail messages coming in the mailboxes of users has grown from a nuisance to a real problem. Some sites report that less than 10 percent of their messages are good. Of course the problem has been transferred to the site administrator who has the control of the mail-server. The most expert user can cope with a self-made filter (procmail) and with the filter embedded in the Mail User Agent application (such as Outlook and Thunderbird).

The problem is not only in terms of time lost by users deleting spam messages, but also the resources needed in the mail-server, and the relation with security problems like viruses and worms used to send spam, scam, phishing and brute force address harvesting.

This is the background that led to the birth of a group of large site (thousand of mailboxes each) administrators in the GARR network. The group goals include the study of all security problems related to electronic mail with a clear urgency to deal with the spam explosion.

The Common Base: SpamAssassin

The group activity concentrated itself in understanding how it would be possible to improve the SpamAssassin (SA) efficiency by reducing negatives and (mainly) false positives.

SpamAssassin is based on heuristic tests using genetic algorithms, and automatic Bayesian statistical corrections. So, the spam nature of every single E-Mail message is determined in a statistical way. Thus, independently of how well tuned the configuration parameters might be, there will always be some positives (good messages erroneously tagged as spam) as well as some negatives (spam messages undetected).

When a message's "SpamAssassin score" is higher than a specific value, the message will be considered spam, and the administrator can decide its destiny: remove it, move it to a specific folder or, most commonly, just flag the E-Mail as probable spam by modifying the subject in order to leave its destiny in the users' hands.

A trivial method used to increase the number of messages detected as spam is either to decrease the spam score level, or increase the score value of some tests. This must be done carefully because, in the first case one will increase the probability of positives, while in the second case one will unbalance the score value between the hundreds of tests.

For these reasons, we decided to study independent methods to improve the filter efficiency, that is, increase the spam/ham distribution separation.

The Bayesian Classifier

The use of Bayesian filters is very effective. With this method, the filter learns the right rule from lists of "certain" spam and ham messages which have been classified by a human. The filter produces two tables of the most frequent words (actually tokens, since the message header is also analyzed) found both in spam and ham messages. After this learning phase, every incoming message is given a score, whose value is assigned depending on the spam/ham token frequency. This method is quite independent from the

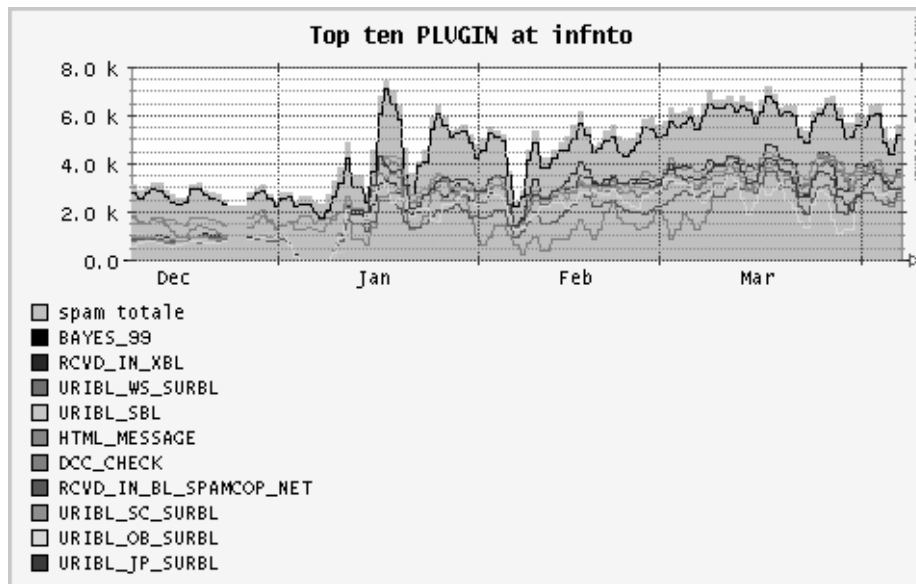


Figure 1: Usual trend for top ten SpamAssassin plug-ins at INFN-TO (Dec 2005-Apr 2006).

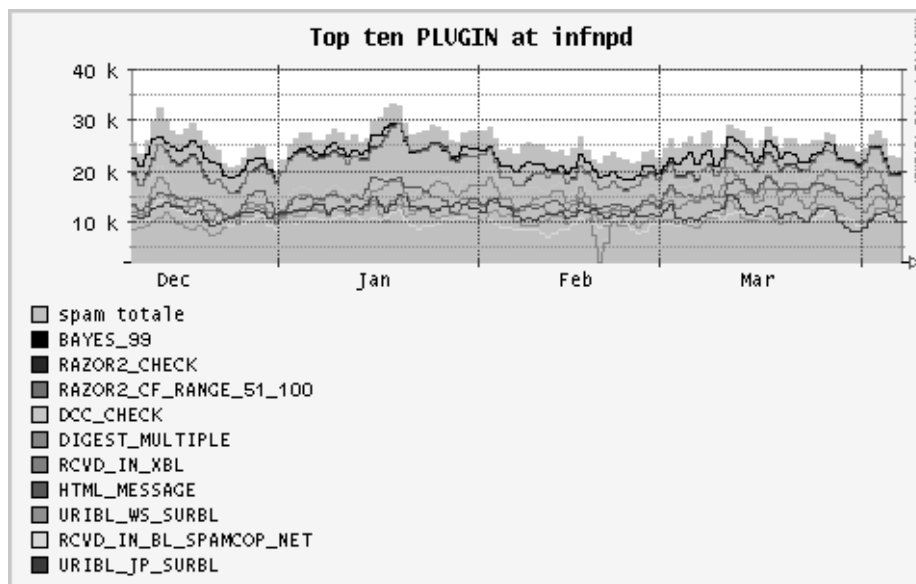


Figure 2: Usual trend for top ten SpamAssassin plug-ins at INFN-PD (Dec 2005-Apr 2006).

rule-driven one, because it also works on good messages and is customized for each mail-server target.

However, this method must be used carefully. The training of the bayesian classifier is a critical step. The efficacy of the bayesian filter is influenced by the spam/ham database size. It must be big enough to contain all relevant tokens, but not so big as to contain old (no longer relevant) tokens. Tests made in the Turin site gave the best results by lowering the parameter `bayes_expiry_max_db_size` from 200 to 100.

The auto-learning option can be exploited by spammers who send E-Mail messages designed to poison the DB with a huge amount of random “non spam” words. In this way, the Bayesian filter might even auto-poison its own DB.

The countermeasure is the DB correction, made by users’ feedback. Every day, each user can re-classify all received positives and also negatives. Tests made in the Turin site have shown that the “right” DB strongly helps in this reclassification of spam/ham messages. For example, in Turin the best results have been achieved with a central DB, so that the spam is targeted on a whole site basis instead of a per-user basis.

The training of the Bayesian filter on the users feedback gave excellent results. As shown in Figure 1, the Bayesian solid test line for the BAYES_99 (99% probability of being a spam) is very close to the total spam detected line.

At the Padua site the above reclassification is made by the site administrator by manually feeding the DB with selected ham & spam. In this case the two lines are a little bit more separated (see Figure 2). Results are even worse at other sites that still haven’t applied any kind of statistical correction.

We’ve set up a monitoring system for the efficiency of the several tests used by SpamAssassin (see Figure 3 and Table 1, referred to one site only). Table 1 represents, for each plug-in:

- score: the single plug-in score assigned by SpamAssassin;
- score %: plug-in score percentage compared with the required score;
- hit: the number of E-Mail messages tagged as spam;
- hit %: percentage of E-Mail messages tagged as spam by the plug-in;

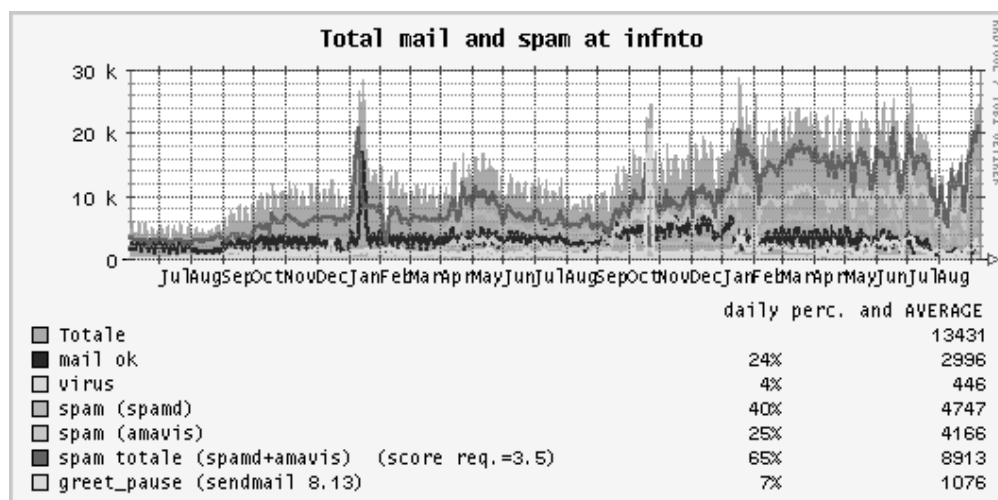


Figure 3: Number of E-Mail messages received at INFN-TO from June 2004 to September 2006.

Plug-in name	Score	Score %	Hit	Hit %
BAYES_99	4.070	116%	5640	94.0%
RCVD_IN_XBL	3.897	111%	4163	69.4%
URIBL_WS_SURBL	2.140	61%	3725	62.1%
URIBL_SBL	1.639	46%	3534	58.9%
HTML_MESSAGE	0.001	0%	3244	54.1%
DCC_CHECK	3.500	100%	3047	50.8%
RCVD_IN_BL_SPAMCOP_NET	3.500	100%	3040	50.7%
URIBL_SC_SURBL	4.498	128%	3035	50.6%
URIBL_OB_SURBL	3.008	85%	3023	50.4%
URIBL_JP_SURBL	4.087	116%	2929	48.8%

Table 1: Numeric values at INFN-TO referred to the 2006-04-07 log.

Switching SpamAssassin from release 2 to release 3 dramatically improved the spam detection ability of the filter, especially with releases 3.1.x. The Bologna site experienced a false negative reduction from 20% to 1%, and a false positive reduction from 0.8% to 0.2%.

Non-Standard Rules

In addition to the standard rules, SpamAssassin can use non-standard plug-ins to further improve its spam detection ability. The Florence site experimented with both the URIBL and SARE families of plug-ins (see Figure 4).

The URIBL family (a sort of distributed blacklist, included by default in the SpamAssassin package, starting rel. 3) is based on the harvest of sites linked from URL's internal to spam messages. Because in most cases, while message headers don't indicate where spam actually comes from, the message body must contain links useful for the spammers. We have found it very useful, especially with SpamAssassin rel.2, where it wasn't included by default.

The SARE family (still not standard) gathers different countermeasures against brand-new spamming techniques. For example, we found the gibberish plug-in (against random words inside the body of some spam messages) to be very useful.

Among non-standard plug-ins we also considered mail-scanners, which are high-performance and reliable interfaces between mail transport agents (MTA) and one or more content checkers. They perform a light anti-spam scan before the final call at SpamAssassin.

Using Mail-Scanners

Special behaviors may be obtained by using the aforementioned mail-scanners. Since the beginning of the group effort, the Turin site has been using the AMaViS [AMA] mail-scanner for both anti-virus and anti-spam filters. The Florence site, instead, developed its own mail-scanner (RJSPAM) [RJS]. However, both

sites use these mail-scanners to reject spam depending on the results of SpamAssassin tests: in Turin for those users that explicitly asked for this behavior (*opt-in*), while in Florence for those users that still haven't explicitly refused it (*opt-out*).

When one receives hundreds of spam messages per day, hijacking them into special folders is completely useless: one will never check them! So, rejecting them might be a wise solution because a spam message is a sender mail-server object, not of the receiver server!

Another positive effect is in terms of performance. With mail-scanners, the scanning takes about a tenth of the time of the usual anti-virus and anti-spam software, since it interacts directly with the libraries, without calling any (slow) executables.

Finally, we suspect there is a useful side-effect: rejecting spam will cause one's (spammed) address to be removed from the spammers' mailing lists! But this behavior has yet to be confirmed (we need further collecting time).

However, the working group members didn't come to a common agreement on the policy of rejecting such messages. Some members think it is unethical, even though it respects all RFC's. Others, instead, simply don't like it.

RBL

Another independent tagging system is the one based on the RealTime Block List [RBL]. The sender E-Mail address is compared with a notorious spammers' database by a DNS query. These databases are maintained by user communities that collect the names of Internet Service Providers housing spammers, allowing open-relays, badly configured proxies or message sending from dynamic addresses. However, blocking lists must be used carefully as they are sometimes too slow in removing good sites, or not very accurate in checking that the reported site was actually guilty. For this reason, RBL scores are better used in conjunction with other rules, and not by themselves.

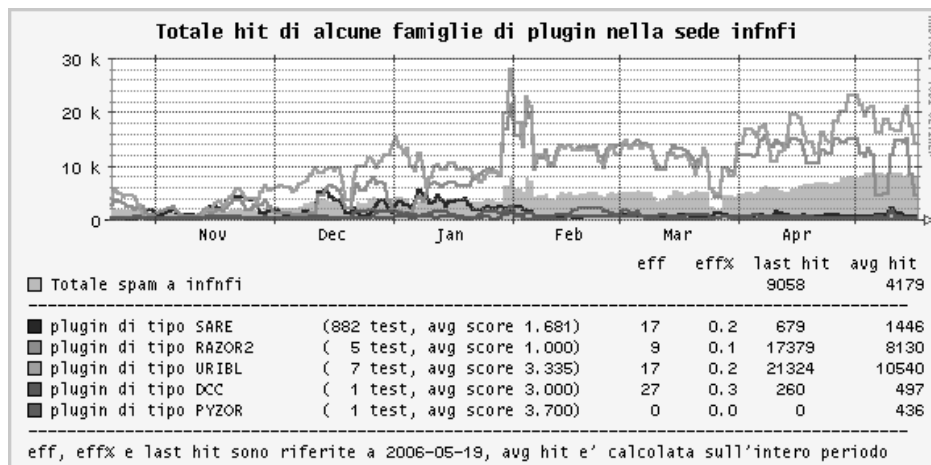


Figure 4: Usual trend for some SpamAssassin plug-in families at INFN-FI (Oct 2005-May 2006).

As stated above, among the several RBL's we have tested, there was a specific one we have found really useful: the URIBL family.

Automatic Categorizers (Community Based & Checksumming)

These systems are based on distributed servers that "count" the received messages in order to compute the probability they may be of type Unsolicited Bulk E-Mail (UBE).

Razor

Razor [RAZ] is based on a database of spam E-Mail messages supplied by humans. A sophisticated scoring mechanism based on correct reports or revocations of spam makes this reporting mechanism unalterable by spammers. Since Razor was based on a private protocol, it wasn't easy to enter into the razor collaborative network. However, RAZOR is partially free.

Pyzor

Pyzor [PYZ] is an open-source rewriting of Razor. Our group is trying to enter its more open collaborative network.

DCC

This product [DCC] is based on a slightly different mechanism. DCC servers automatically "count" bulk E-Mail messages by trying to cut away the variable elements and keeping the fixed ones, generating a checksum for each of them. After that, DCC servers exchange this information with a flooding mechanism. In this way, for every incoming message, a mail-server can ask a DCC server for the probability that this message, with such a checksum, has to actually be UBE. Any DCC server gives answers to both client types, anonymous and registered. Higher priority is usually given to the latter ones.

DCC is an open system where new servers are always welcome. All working group member sites have been using DCC clients since the group was founded, while four sites are even hosting servers. The first Italian DCC server was installed at INAF in Palermo, immediately followed by the INFN site of Turin [DCT]; then Rome and Bari. At first, we thought three servers would be enough, but new GARR site volunteer servers have always been welcome in order to better serve the increasing number of GARR clients. Our achievement is an improvement of the method efficiency, both by reducing the response time

and by increasing DB data about our *domestic* spam. Even though the service is offered on a best effort basis only, the Turin server has been recently replaced with a more powerful one; due to this refurbishment, it has been included in the default DCC server alias (*dcc*.dcc-servers.net*) and is currently serving thousands of clients worldwide, checking an average of 15 M messages/day (12.5% of the total, see Figure 5).

After using this new DCC server, the Turin mail-server unexpectedly started increasing its spam detection efficiency, with false positives and false negatives nearly equal to zero. We are investigating this coincidence, but we suspect that the different new role of the Turin DCC server affected its spam detection ability (see Figure 6). In fact, because of its inclusion in the default DCC server alias, its database now contains data which is more up to date than when it was interacting with fewer clients (the *domestic* ones only) and processing fewer messages, and the DB was aware of the rest of the world by a data flooding between DCC servers only.

DCC Reputations are a distinct mechanism based on and contributing to DCC data. In part to minimize abuse by anonymous users, DCC Reputations are available only in the commercial version of the DCC software. For this reason, our working group still hasn't considered implementing it, but we cannot exclude begging it from Vernon in the future ...

DSPam

DSPam [DSP] is a spam detection system proposed as an alternative to SpamAssassin. It's based on highly sophisticated statistical techniques only. Even though the authors achieve an efficiency better than 99.9%, not only did our testing not reach this value, but showed results worse than SpamAssassin. Perhaps our training phase quality (not accustomed to purely statistical methods) was not as good as required. For the future, we are currently considering its possible implementation as a plug-in of SpamAssassin.

Viruses and Worms

Fortunately, nowadays anti-virus filters for mail-servers are quite robust and reliable. Free software surely has good examples of well designed products (i.e., ClamAV), but commercial software is usually better due to faster virus definition updates (i.e., Sophos AV). Nevertheless, commercial software doesn't always imply more reliability.

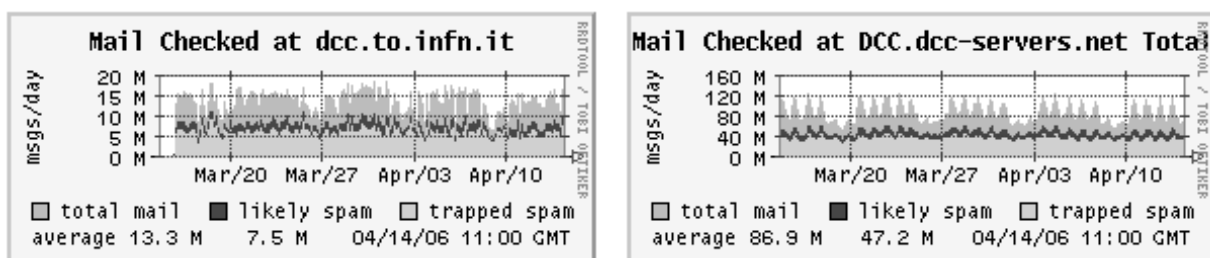


Figure 5: E-Mail checked at dcc.to.infn.it, compared with the total at dcc.dcc-servers.net.

For the whole working group the number of viruses *revealed* by E-Mail decreased very quickly by simply adopting good anti-virus filters, implementing greylisting, and following some best practices (discussed in the next section).

As shown by the green line in Figure 7, in less than one year, the total amount of viruses *revealed* in some GARR sites decreased from 30% to nearly zero.

Best Practices

An important part of the working group activity has been dedicated to “best practices,” that is *suggestions* to improve the security of E-Mail services.

Among the most important points identified:

- Edge routers should allow incoming traffic through port 25 (smtp) to reach only the

domain official mail-server, in order to prevent generic LAN computers from being used as mail-relay.

- Edge routers should pass only outbound traffic through port 25 (smtp) from the domain official mail-server, in order to prevent viruses and worms from sending E-Mail messages (at present, a typical behavior). Ports 587 (msa – mail message submission), and possibly 465 (formerly for Windows Outlook mail message submission, currently deprecated), must be left *open*, so *roaming* users can use their own MTA from the exterior of the LAN.
- Roaming users must be able to use their own MTA from the exterior of the LAN by authentication only. This way one can implement sender control methods (i.e., SPF). Be careful

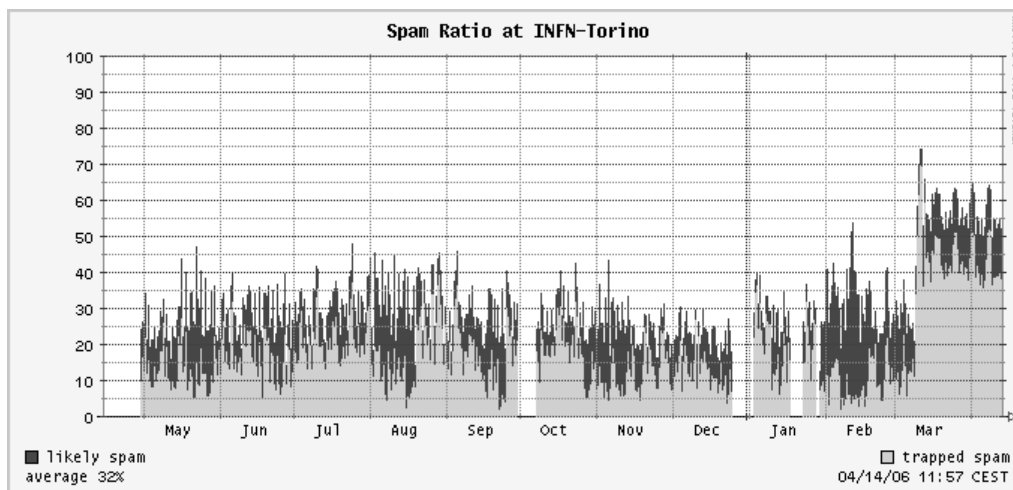


Figure 6: Spam ratio at dcc.to.infn.it from May 2005 to April 2006.

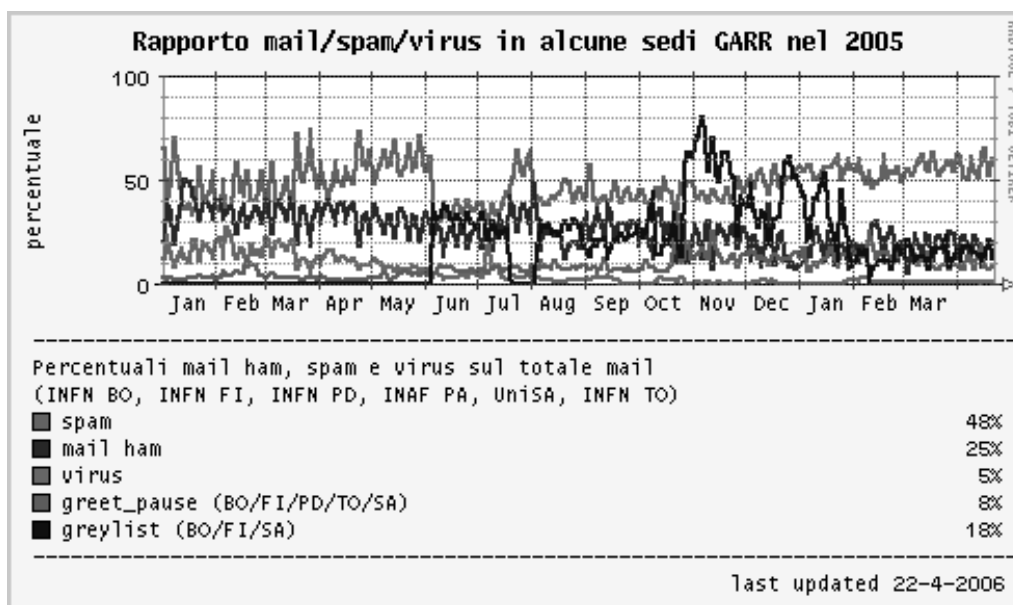


Figure 7: Ham, Spam & Virus on total E-Mail, in some GARR sites from Jan 2005 to Apr 2006.

when using authentication together with SpamAssassin! If a user's Internet Service Provider (ISP) has a bad reputation, SpamAssassin may consider his messages to be spam. In order to prevent this, the following rule should be added when using Sendmail as MTA:

```
header LOCAL_CERT_GNORRI_ON Received =~
/^HEADERSTRING\].+verify=OK\)/
score LOCAL_CERT_GNORRI_ON -15
where HEADERSTRING is typical of one's
mail-server header.
```

ISP's with bad reputation are a very annoying problem. Unfortunately, in Italy it's a common issue, mainly with cheaper and faster ISP's. So, due to the latter ones, non guilty users risk being classified as spammers even when using web-mail interfaces, if they list the ISP in the message header because of some RFC (821) interpretation.

- The anti-virus software installed on the mail-server should be configured to prevent itself from sending information messages (about infected E-Mail messages) to the senders, since these ones are usually *spoofed* (false).
- The Sendmail (v. 8.13) Greet Pause feature should be kept active. Through this mechanism, the SMTP connection is rejected if the sender mail-server doesn't wait for the "220" greeting answer. This is the typical behavior of spammer software and viruses, which are usually not able either to hold-on or, after a time-out, to retry the connection as *regular* MTA's do.

For (not only) GARR users, an *Installation manual for an electronic mail service with anti-virus and anti-spam filters* [MAN] has been published (presently available in Italian only).

Sender Domain Authentication

Another important argument is the sender server authentication, because partially related to the spam problem (many spam messages come with the sender spoofed). Moreover, roaming users must be able to use their own MTA from the exterior of the LAN by authentication only.

Several methodologies have been proposed, even though not yet merged into a RFC. Among the several proposals, two different technologies have emerged. The first one is known as Sender Policy Framework [SPF], the second as Sender-ID [SID], and both aren't directly used for fighting spam, but only for authenticating the mail-server that is sending an E-Mail message. The basic concept of these systems is that the user of a generic domain can send messages only by those mail-servers explicitly authorized by their own domain. This mechanism prevents E-Mail messages with a spoofed sender address from being sent, but mainly prevents infected computers from sending spam or viruses.

Our working group chose to test SPF, verifying all possible benefits. A site can become SPF compliant by publishing the mail-server names authorized to send E-Mail messages with this site sender addresses. The best achievements would be reached whenever the majority of the sites publish the list of the authorized servers as well as enter them into their own DNS records. However, we still are far from this status. However, since some big ISP's already publish SPF records, it's possible to use this information in order to modify the SpamAssassin scoring. It shouldn't be forgotten that ISP's implementing SPF must inform their roaming users that they cannot sign their outgoing E-Mail messages with their own domain name when using *external* IP addresses, because the SPF check would fail. Thus, before publishing one's SPF record, one's users should be aware they can send E-Mail messages by using authorized mail-servers only. For this reason, relaying must be allowed but only by using specific authorization mechanisms (i.e., either via password or X.509 Certificate).

Our tests, made on an actual domain (University of Salerno), showed that 12% of messages received in one month (650K) came from senders whose mail-servers were already SPF compliant. To this value may be added another 20% of messages belonging to the examined domain, reaching therefore a considerable 32% of E-Mail messages carrying SPF information.

Starting June 2006, we decided to enable SPF on all mail-servers managed from our working group members. At present, only in a *soft* way, just to test it by the related SpamAssassin rules.

Greylisting Technologies

Nowadays over 60% of spam and viruses come from infected computers (hijacked computers) and not from real servers. Machines infected by these viruses try to emulate the behavior of a full-fledged mail-server, but this imitation fails to implement some functionalities. This lack of functionality can be used to differentiate a real mail-server from an infected machine.

The functionality used to differentiate between real servers and infected machines is the retransmission capability, that is the capability of a real server to retransmit a message if a receiver server couldn't (or wouldn't) receive E-Mail from another server (e.g., the receiver server is overloaded, or the receiver server uses Greylisting).

The Greylisting [GRE] method is very simple. It examines only three pieces of information (which we will henceforth refer to as a "triplet") in any particular E-Mail message delivery attempt:

1. The IP address of the host attempting the delivery
2. The envelope sender address
3. The envelope recipient address

From these, we now have a unique triplet for identifying an E-Mail "relationship." With this data, we simply follow a basic rule, which is:

- If we have never seen this triplet before, then refuse this delivery and any others that may come within a certain period of time with a temporary failure.

Since SMTP is considered an unreliable transport, the possibility of temporary failures is built into the core spec (see RFC 2821). As such, any well behaved message transfer agent (MTA) should attempt retries if given an appropriate temporary failure code for a delivery attempt.

The main two limits of this approach are:

- the delay time due to the temporary failure (from 30 minutes to some days);
- some ISP, like hotmail, use an entire subnet for SMTP retransmission and not a single IP thus implying more difficulty to identify the triplet.

Although the first one of these problems could be the less important (since SMTP protocol does not guarantee the delivery time for an E-Mail message), it becomes very annoying to the community of users when the E-Mail Service is used more and more like an Instant Message application rather than a postal service.

The second problem can be easily solved using the full sub range ("C" class Network) of an ISP as the IP address. This way every server in the network that will retry to forward the E-Mail message previously interrupted will be recognized as being the same server.

There is also a second approach to the subnet problem: SPF. Domains like Hotmail and AOL, which are frequently faked and abused by spammers, have introduced an SPF record into their own DNS domains. That record can be used to improve the behavior of greylisting. If a receiving server is with greylisting enabled and receives an E-Mail message from Hotmail or AOL, it could look up the SPF record of the sender domain and compare the sender IP with the ones allowed for the sender domain. If the sender address is allowed then the greylisting should accept

that message as it is a real message. Spam and viruses which are sent from hijacked computers don't come from the authorized servers but rather from the infected computer itself.

The Advantages

Sharp Reduction of Received Viruses and Spam

Greylisting reduces the number of accepted E-Mail messages processed by the receiver server and this greatly reduces the amount of spam and virus to check. The price is a rise of the average delivery time and an increase in E-Mail traffic (due to the retransmissions).

In Figure 8 one can notice that the traffic shape changed enormously in the month of June 2005, in concurrence with activation of Greylisting filter at INFN-FI. The above picture represents:

- Red line: messages identified as spam by SpamAssassin (with a score threshold of 3.5);
- Cyan line: messages rejected by greet pause of Sendmail 13.x;
- Black line: messages rejected by (classic) Greylisting;
- Yellow line: messages containing viruses;
- Blue line: "clean" messages;
- The last, the grey contour, is the sum of all the contributions.

Two important aspects to underline are the trends of the blue and grey lines: the first shows that real E-Mail preserves the same behavior, the second represents the total number of messages processed by the servers. The jump from 15K messages to over 30K daily is due to retransmissions.

Sensitive Reduction of the CPU Load

Some measurements taken at University of Salerno, prove that only 20% of the incoming E-Mail is actually re-forwarded. About 80% of E-Mail is immediately blocked by the Greylist algorithm. Considering moreover that Greylist's algorithm is activated only during the first step of the E-Mail protocol

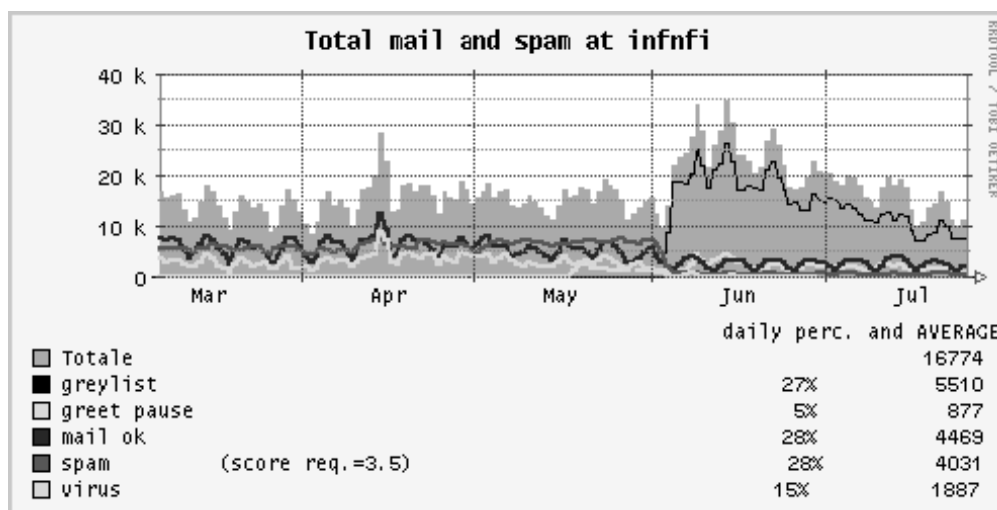


Figure 8: Number of E-Mail messages received at INFN-FI from March to July 2005.

(header transmission), it means that 80% the incoming E-Mail is zero-cost rejected as can be seen from Figures 9 & 10.

In the first graph (Figure 9) we exhibit the CPU load of a mail-server (mx3.unisa.it) equipped with greylisting technology while in the second graph (Figure 10) we are reporting the CPU load of another server (mx2.unisa.it) without greylisting functionalities. As can be seen the load in the first graph is only a small portion of the CPU utilized in the second situation.

The Disadvantages

The Delivery Time of the First E-Mail Message

The request for comment 2821 [RFC 2821], paragraph 4.5.4.1 Sending Strategy, asserts that a sender must resend a rejected E-Mail message after at least 30 minutes:

“The sender MUST delay retrying to particular destination after one attempt has failed. In general, the retry interval SHOULD be at least 30 minutes; however, more sophisticated and variable strategies will be beneficial when the SMTP client can determine the reason for non-delivery.”

Moreover it would have subsequently executed two connection attempts in the first hour and one every two or three hours thereafter:

“Experience suggests that failures are typically transient (the target system or its connection has crashed), favoring a policy of two connection attempts in the first hour the message is in the queue, and then backing off to one every two or three hours.”

Other measures (Figure 11) performed on our servers at the University of Salerno, show that approximately 30% of E-Mail is resent within approximately 10 minutes, 60-70% of incoming E-Mail is instead delivered after 30 minutes and finally, 80-90% of E-Mail is delivered within 60 minutes from the first attempt.

There is also another disadvantage that can be felt as particularly annoying: websites that require you to create an account and confirm your E-Mail address before you can begin using them. Due to the fact that greylisting will delay the initial E-Mail message containing your signup confirmation link (maybe for some minutes or perhaps some hours), it will introduce a waiting period even though the actual website may send out your E-Mail confirmation code immediately.

A Better Greylisting

A Home-made Experimental Technology

The new approach to Greylisting is the union of classic Greylisting with a spam filter. While Greylisting tells us *if a server is RFC compliant*, spam filter tells us *at which level of confidence a message can be considered ham or spam*. Joining the two algorithms allows some optimization like bypassing the Greylisting algorithm if the E-Mail message has a very low score or, in the case of multiple recipients, accept an E-Mail message if only a single well-know triplet that “introduce” the sender for all the others exists.

These optimizations aim to reduce the delivery time for the first E-Mail message.

The Achieved Result

As can be seen from Figure 12, more than 40% of E-Mail messages are delivered instantaneously. Approximately 80% of them are instead delivered

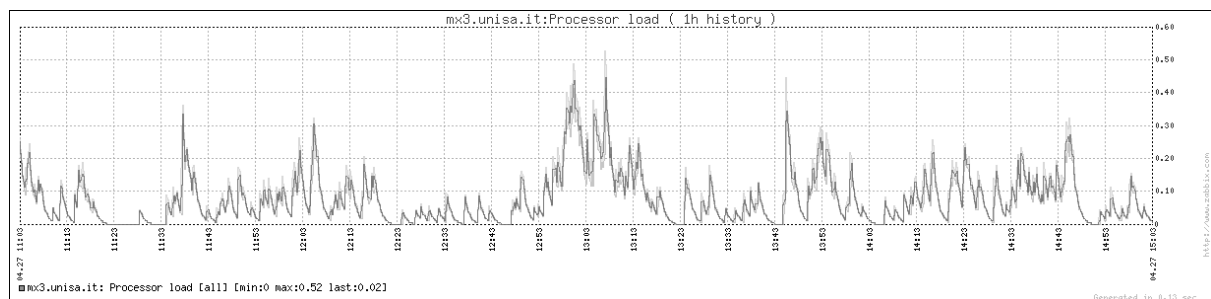


Figure 9: CPU load of a mail-server with classic greylisting at UNISA.



Figure 10: CPU load of a mail-server without any greylisting at UNISA.

within 30 minutes and 90%, with peaks of 100%, within 60 minutes. Essentially, from a direct comparison with the previous diagram, one can notice a sharp improvement on the accepted times.

As shown in Figure 13, in one hour the mail-servers of Università degli Studi di Salerno received 1197 E-Mail messages. Only 12% of them are accepted as ham, while 3% are instead classified as SPAM and 0.8% of E-mail is infected. The remaining part, 84%, is rejected. Practically, 16.5% is rejected thanks to Greeting Pause, 21% is rejected due to non-existent recipients and 45% is rejected by greylisting where only 9.8% of these messages are retransmitted and, therefore, accepted.

The Cost

The consequence of this result is a sensible growth in the load on the server and in the used network band. This is due to the fact that all the E-Mail messages are, at least once, processed from the anti-spam & anti-virus filter (see Figure 14).

In Figure 14, classic greylisting has been tested from October 2005 until January 2006, while experimental greylisting has been running since February 2006.

Conclusions (The Lesson Learned ...)

A huge amount of graphs and statistics have been produced and are available in (almost) real-time

on the working group web site and wiki [WSM]. They significantly helped us in highlighting many of the achieved results, or just to quickly understand *why the devil that stupid mail-server stopped working* ...

Note that all the produced software is freely downloadable from the sites listed either in the above paragraphs, or in the “Bibliography” section. This is the real-life story of 10 guys that started “playing” within a new working group, and finished working seriously, giving valuable results to the Italian Academic (and non) network. From Figures 15 & 16 we see that, in spite of a more or less constant increase of spam messages, we still have our mailboxes clean thanks to a constant fighting activity of the whole working group. Unfortunately, new/original tools are not always useful to get good results and with our work we have demonstrated that one can achieve excellent results even by using just standard tools. We’ve seen that the more SpamAssassin plug-ins one uses, the more efficiency one will obtain.

The statistical (Bayesian) approach resulted very effective (> 90% hits), and spammers seem still unable to get around it.

Provided you consider it ethical, rejecting spam let’s the users save time in checking the specific folders where spam messages are hijacked.

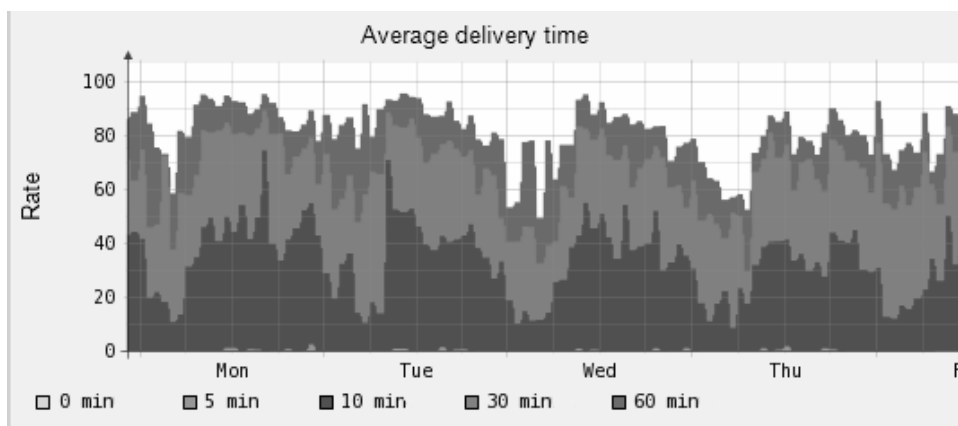


Figure 11: Average delivery time with classic greylisting at UNISA.

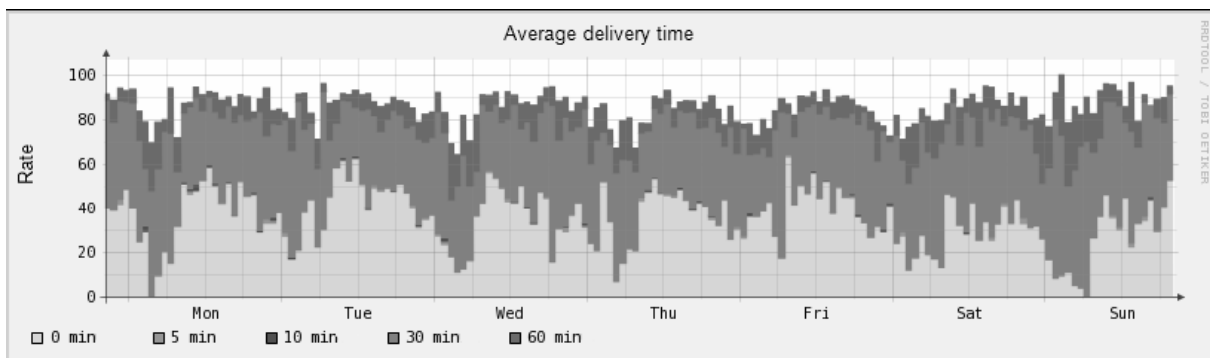


Figure 12: Average delivery time with experimental greylisting at UNISA.

Our very recent Italian DCC Servers Network has surely improved the spam detection efficacy of the Italian (and non) clients installed on mail-servers.

We found that greylisting provided a significant reduction in the amount of spam received and processed and that greylisting combined with SPF checking provided the greatest benefit.

Last, but not least, without a minimum of best practices, the best anti-spam packages might result completely useless.

Thus, without this constant battle, is the war lost? Will I ever be able to run a software that lets me forget my mail-server?

Author Biographies

Roberto Cecchini: Degree in Physics; Computing Centre Coordinator at INFN in Florence; since 1999 GARR IT security service (GARR-CERT) Coordinator; since 1998 INFN Certification Authority (INFN-CA) Coordinator.

Fulvia Costa: LAN Manager & APM at INFN in Padua.

Alberto D'Ambrosio: Electronic Technician; Computing Centre System Administrator at INFN (from 2000 at Turin Section, from 1992 to 2000 at Gran Sasso National Laboratories); previously analyst/programmer in the area of industrial automation.

Domenico Diacono: Computing Centre System Administrator at INFN in Bari.

Giacomo Fazio: IT Engineer; System and Network Administrator at INAF/CNR in Palermo; E-Mail system manager, Computing Centre Coordinator (from 1991 at IFCAI, then IASF, now INAF); previously programmer for research groups working in the area of astrophysics.

Antonio Forte: IT Technician; Computing Centre System Administrator at INFN in Rome1 (from 1/1/2004), Turin (1998-2003) & Rome2 (1996-1998).

Matteo Genghini: Electronic Engineer; since 2002 Computing Centre IT Coordinator at IASF/CNR in Bologna; previously multimedia programmer.

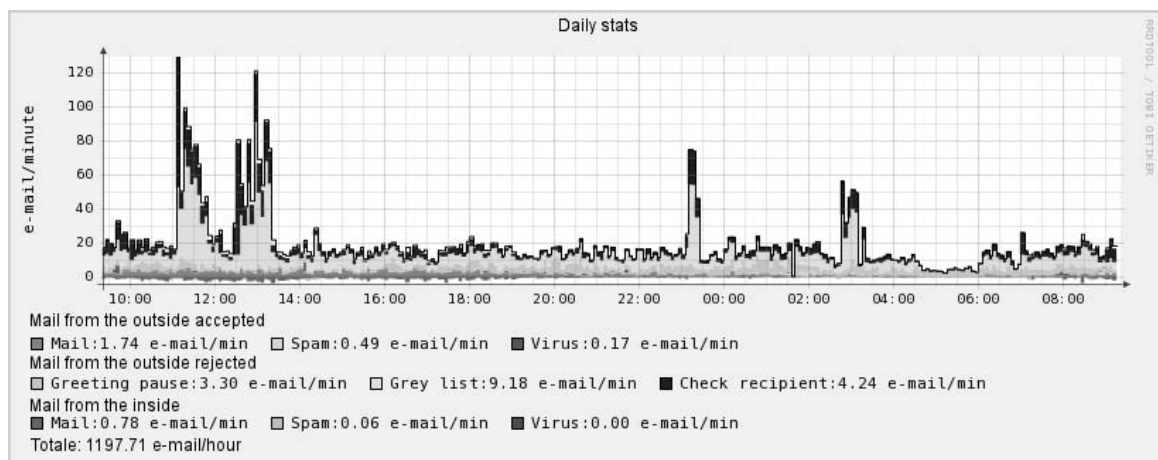


Figure 13: Daily E-Mail stats at UNISA.

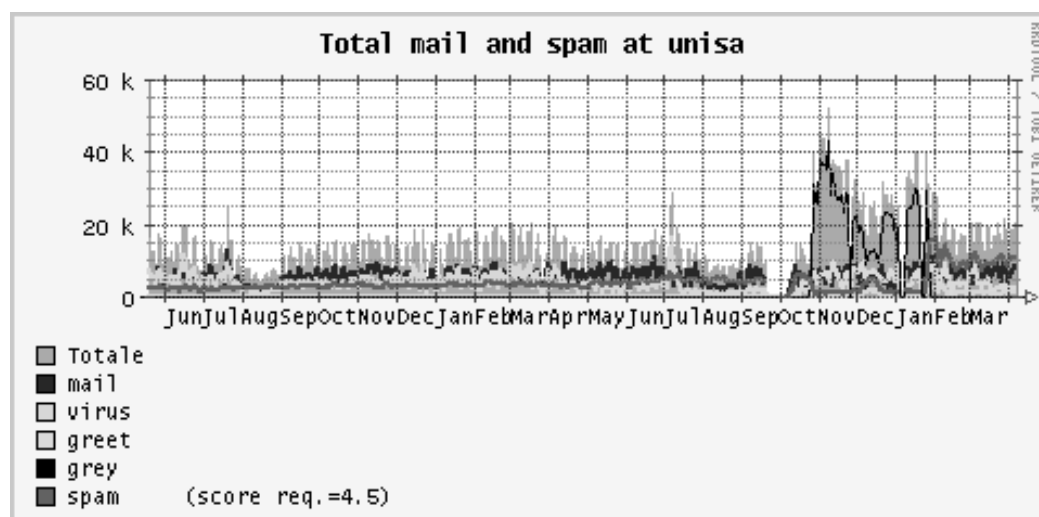


Figure 14: Number of E-Mail messages received at UNISA from May 2004 to April 2006.

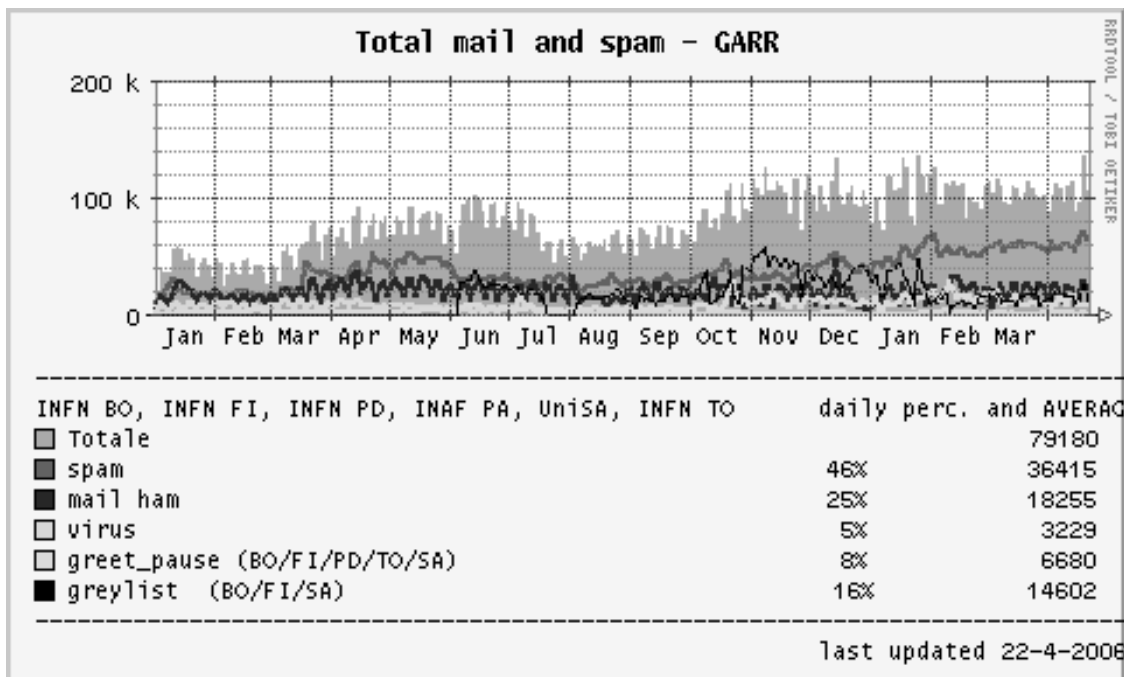


Figure 15: Number of E-Mail messages received at some GARR sites from Jan 2005 to Apr 2006.

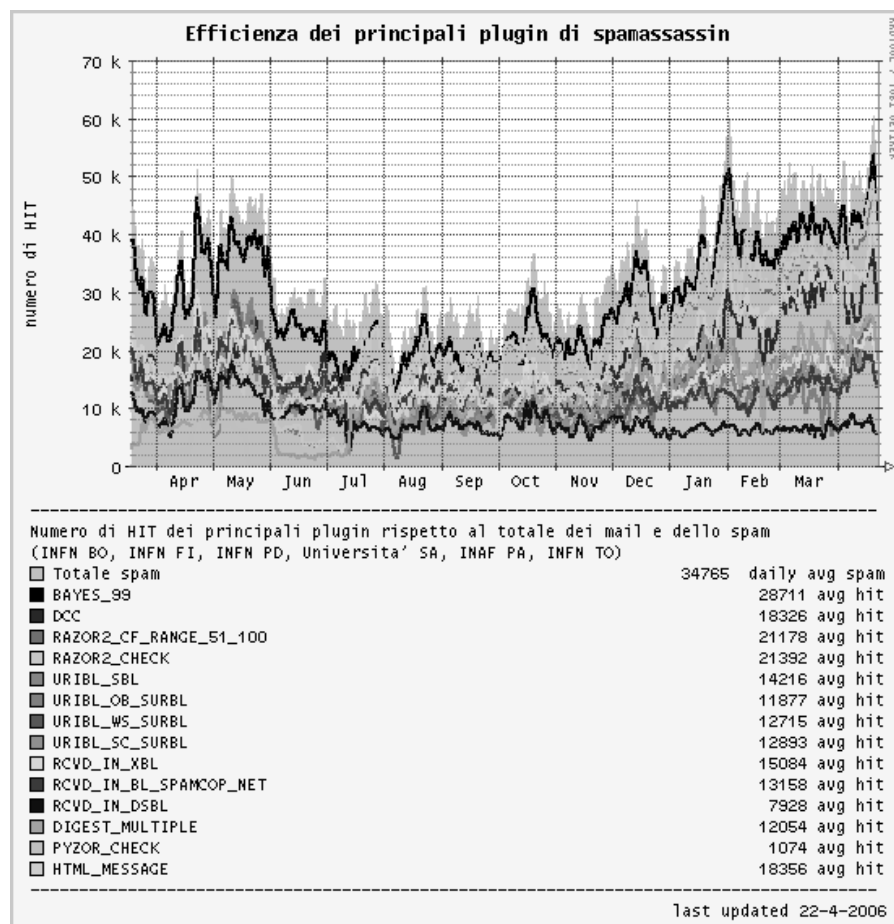


Figure 16: SpamAssassin plug-ins at some GARR sites from March 2005 to April 2006.

Michele Michelotto: Physicist; since 1997 Computing Centre Coordinator at INFN in Padua; previously, offline analysis in High Energy Physics experiments at CERN (Geneva) and INFN (Padua & Legnaro).

Ombretta Pinazza: Degree in Physics; since 1998 IT expert at INFN in Bologna; previously, Researcher at the Engineering Faculty in Bologna and Researcher at TESRE-CNR in Bologna.

Alfonso Sparano: Degree in IT Engineering; Data Processing Centre IT services administrator at the University of Salerno.

Bibliography

- [AMA] <http://www.ijis.si/software/amavisd/>.
- [DCC] <http://www.rhyolite.com/anti-spam/dcc/>.
- [DCT] <http://www.to.infn.it/dcc/>.
- [DSP] <http://www.nuclearelephant.com/projects/dspam/>.
- [GRE] <http://www.greylisting.org/>.
- [MAN] Bar, Giorgio, Alberto D'Ambrosio, Franca De Giovanni, "Manuale di installazione di un servizio di posta elettronica completo di filtri anti-virus e anti-spam," *Installation manual for an electronic mail service with anti-virus and anti-spam filters*, INFN/TC-05/09, SIS-Pubblicazioni, Frascati, Rome, Italy, <http://www.lnf.infn.it/sis/preprint/pdf/INFN-TC-05-9.pdf>.
- [PYZ] <http://pyzor.sourceforge.net/>.
- [RBL] <http://www.webopedia.com/TERM/R/RBL.html>.
- [RAZ] <http://razor.sourceforge.net/>.
- [RJS] <http://mips.df.unibo.it/sw/rjspam0.1.tar.gz>.
- [SID] <http://www.senderid.org/>, <http://www.microsoft.com/mscorp/twc/privacy/spam/senderid/default.aspx>.
- [SPF] <http://spf.pobox.com/>.
- [WSM] <http://www.garr.it/WG/sec-mail/>, <http://secmail.unisa.it/>.

Glossary

- Spam:** E-Mail messages we wouldn't like to receive
- Ham:** Good (non Spam) E-Mail messages.
- UCE:** Unsolicited Commercial E-Mail. Commercial E-Mail messages not requested, thus not welcome, sent to one recipient or thousands.
- UBE:** Unsolicited Bulk E-Mail. E-Mail messages sent to thousands of recipients. Not necessarily commercials; might even be sent to check the actual existence of the recipients.
- False Negatives:** Spam messages erroneously detected as Ham.
- False Positives:** Ham messages erroneously detected as Spam.

A Forensic Analysis of a Distributed Two-Stage Web-Based Spam Attack

Daniel V. Klein – LoneWolf Systems

ABSTRACT

Open mail relays have long been vilified as one of the key vectors for spam, and today – thanks to education and the blocking efforts of open relay databases (ORDBs) – relatively few open relays remain to serve spammers. Yet a critical and widespread vulnerability remains in an as-yet unaddressed arena: web-based email forms. This paper describes the effects of a distributed proxy attack on a vulnerable email form, and proposes easy-to-implement solutions to an endemic problem. Based on forensic evidence, we observed a well-designed and intelligently implemented spam network, consisting of large number of compromised intermediaries that receive instructions from an effectively untraceable source, and which attack vulnerable CGI forms. We also observe that although the problem can be easily mitigated, it will only get worse before it gets better: the vast majority of freely available email scripts all suffer from the same vulnerability; the load on most proxies is relatively very low and hard to detect; and many sites exploited by the compromised proxy machines may never notice that they have been attacked.

Introduction

As new defenses against spam are developed, spammers have been forced get more inventive in their attacks. Much as antibiotic resistant bacteria force the development of new drugs, so do spam-resistant mail servers drive the spammers to continually create new methods of disseminating their advertisements. A modern anti-spam arsenal provides defense with a combination of Bayesian filters, keyword matching, fuzzy checksums, header and source IP checks.

Spammers have responded with text that defeats filters, omits key words with unusual spelling or use of graphics, random strings to confound checksums, and more and more legitimate-looking headers. The one thing that can truly defeat spammer is to deny them the means to send their email. Since recent legislation has made it undesirable to have a traceable source IP address (since getting caught can now mean stiff fines and/or jail time¹), spammers used open relays in foreign countries as an indirect (and difficult to trace) means of promulgating their wares.

Indeed, for a while it was thought that closing off open relays and/or denying them the ability to send mail might be a solution. Although it took time to spread, updates to the default configurations of mail transfer agents (MTAs) went a long way towards alleviating the open-relay problem, while education through a distributed denial of service (in the form of ORDBs) have made open relays, if not a thing of the past, at least a non-problem.

¹On September 5, 2006 the Virginia Court of Appeals (Record No. 1054-05-4) upheld the felony conviction of Jeremy Jaynes under the state's anti-spamming law (18.2-152.3:1). The sentence (which was affirmed in the appeal) was nine years imprisonment.

However, open relays fall into two classes: MTAs and MUAs (mail user agents), and the ORDBs only deal with the former class. MUAs are often thought of as applications that send mail for an interactive user, but they can also be applications that send mail on behalf of a script. In this case, the application is a CGI script, and an improperly configured CGI script can be exploited to send a payload to locations that were not envisioned by their designers.

Vulnerability Summary

The vulnerability in CGI scripts that send email is easy to spot, and trivial to fix. It may best be summarized in the following highly simplified example (see Example 1).

At first blush, the form and the CGI script look innocuous enough. A surfer can enter a single line of text for their name, another for their email address, and a multi-line complaint. The script then sends an email from the user to a fixed address (in this case, mine), and lets me know their complaint. Compliant browsers all provide this functionality, but the problem is not in browsers, but rather in the fact that it does not take a browser to surf the web! Surely the notion of robots is no mystery (all of the search engines use them to harvest information from the web), so it should likewise be no surprise that a robot can submit forms. Combine this with a fundamental feature of the HTTP and SMTP protocols, and you have a trivial means for a spammer to abuse this simple form.

Although browsers only allow a single line of data to be supplied for an input item of type "text," the HTTP protocol allows arbitrary text, and robot can easily provide that arbitrary text. When connecting to Sendmail (and other MTAs), mail headers are separated

from the body of the message by a blank line. So if a legitimate surfer enters “Dan Klein” for the Who field, “dan@klein.com” for the Addr field, and “This form\nis really dumb!” for the Complaint field, then the following would be handed to Sendmail:

```
To: dvk@lonewolf.com
From: Dan Klein <dan@klein.com>
Subject: Whine whine whine

This form
is really dumb!
(whew)
```

Notice that the newline in the Complaint field is accurately translated when sending the email. So now consider that a spammer has instead supplied the value in Example 2 for the Addr field. Here is what is handed to Sendmail:

```
To: dvk@lonewolf.com
From: Dan Klein <dan@klein.com>
Bcc: your@address.com, target@victim.com,
     another@mark.com, more@suckers.com
Subject: Increase your pennies

Get bigger pennies! The 19th century
English pennies are HUGE! Impress
your woman!

>Subject: Whine whine whine

This form
is really dumb!
(whew)
```

Since the spammer cannot see the script (only its results), s/he must treat it as a black-box, making a number of attempts until the proper formatting is discovered. Notice the careful placement of the ‘>’ character to guarantee (in this case) that Sendmail properly parses the input. The spammer typically creates a short-term-use email account and uses that address as a test target, trying a collection of well-known formats with specially coded messages with an annotation (representing the format type) included. Once one email is

received at the test address, the spammer knows from the annotation the correct format to use.² Suddenly the innocuous email form is a vehicle for sending spam!

In retrospect, I should have noticed the problem with my form when the spammer first tested their script. I received a short collection of very similar spam messages from all over the world, but like most people, I simply deleted them. Initially, I received them because the spammer had not yet determined how to properly exploit my script (for example, they had appended the Bcc addresses in the Complaint field).

After the exploit was fine-tuned to match my script, I received approximately 5,000 spam emails (because in my script, I was listed as the To address). Each of the messages were also Bcc’d to 380-390 other addresses. Had I not detected the attack, this ultimately would have resulted in close to 2,000,000 spam messages being sent from my mail server. It is interesting to note that my desktop spam blocking software helpfully deleted most of the spam messages addressed to me. Thus I never knew I was being attacked – at least until I checked my various RRD logs, as shown below.

The Larger Problem

A very quick survey using Google reveals the severity of the problem. Searching for “email cgi script,” “Perl CGI email” and “Perl email script” yielded the expected plethora of results, but a union of the top-10 of each search yielded 21 unique pages. Of these 21 scripts, one third suffered from the vulnerability described above:

- 7 scripts had obvious flaws that allowed spam email exploitation of the type seen in the example above.

²Some of the test emails also include the correct use of MIME headers and separators, some of which (through *multipart/alternative*) appear to be specifically calculated to hide the original legitimate contents of the form-generated email.

<pre><FORM ACTION=mail.cgi> Your name: <INPUT TYPE=TEXT NAME=Who>
 Your email address: <INPUT TYPE=TEXT NAME=Addr>
 Your problem: <TEXTAREA NAME=Complaint> Type your complaint here! </TEXTAREA> <INPUT TYPE=SUBMIT> </FORM></pre>	<pre>#!/usr/bin/perl use CGI ':all'; import_names; open MAIL " sendmail -oi -t"; print MAIL "<<==END=="; To: dvk@lonewolf.com From: "\$Q::Who" <\$Q::Addr> Subject: Whine whine whine... \$Q::Complaint (whew) . ==END== close MAIL;</pre>
---	--

Example 1: CGI Form and CGI Script.

```
dan@klein.com>\nBcc: your@address.com, target@victim.com,\nanother@mark.com,more@suckers.com\nSubject: Increase your pennies\n\nGet bigger pennies! The 19th century\nEnglish pennies are HUGE! Impress your woman!\n\n
```

Example 2: Nefarious text handed to the complaint field.

- One of the flawed scripts was #1 on a particular Google results page.
- Another of these scripts “sanitized” the user data by removing brackets and shell wildcard characters from mail headers, but failed to remove line breaks!
- A third script removed all instances of “\r\n” followed by removing all “\n” characters. This allows a spammer to use “\n\r” as a line break, leaving a bare “\r” after the so-called sanitization is performed.
- 3 scripts had code to successfully prevent spam email exploitation.
- 2 were at perl.com (and thus were assumed by this author to be correct).
- The remainder of the links were located on pay-sites, were too convoluted to search, or had bad pointers to scripts.

Matt’s Script Archive (a tremendously popular website for perl scripts) provided a secure script, but looking at his change log, I found that the script was only secured in August of 2001. Although he urges current users to upgrade to the secured version, the site also has the following note: “This script has been downloaded over 2 million times since 1997.” I noted that other publicly available scripts were based on the buggy versions of Matt’s script, and hesitate to guess how many of the buggy derivative scripts are still in use throughout the world.

At least half of the most likely-to-be-copied scripts were sufficiently flawed as to allow easy exploitation by a spammer. A random sampling of other email scripts listed later in Google’s results indicated that on the order of half the scripts were equally vulnerable.

Solution

Fortunately, there is a trivial fix to the problem. An obvious (yet regrettably often overlooked) rule of thumb is “never trust your users” – especially when those users are unvetted and/or from the web. A simple rule that the script could follow is: “don’t trust user input,” and simply sanitize the user data to accomplish this. Not allowing any user data in email headers is the best solution – fixed header-content guarantees a known result (using Sendmail’s -oi switch is also highly recommended, or whatever the equivalent is in other MTAs). However, since it is also convenient to be able to reply to emails from forms (and thus allowing the user to specify input that will ultimately be placed in email headers), a less draconian solution is to simply remove newlines (or more generally, replace all contiguous whitespace with a single space character) in any user input that may find its way into an email header. Either action will fix the vulnerability and allow a user to *Reply* to form-based email.

There is no need to go to the extra effort of determining if the user has supplied a valid email address.³

Sendmail will simply fail to deliver mail with bogus From, To, Cc, or Bcc headers, so in general it is better to leave the address parsing to the program most responsible for it.

Discovering the Attack

In addition to reading my email, one of my morning rituals is to scan the SNMP statistics from the various machines on my network. A single click in my tabbed web browser brings all this data readily to hand.⁴ These include critical statistics such as network load, free disk space, web server statistics and email traffic, as well as more incidental data including load average, number of active users and processes, machine uptime, and NTP synchronization data. Any anomalies can thus be quickly investigated, and problems can be readily averted or remedied.

My network consists of my laptops and desktops, a web server, an email server, a backup server, a pair of name servers, and a semi-public wireless network. Figure 1 shows the overall traffic on my T1 serial line. The pattern of activity is not particularly noteworthy, and can readily be accounted for by a newly popular page on one of my websites, or one of my wireless customers uploading a particularly large file.

Figure 2 (available to me on the same dashboard) shows network traffic broken down by LAN. Although a second look showed me that the primary load was on the downstairs network (where the mail and DNS servers are located), initially this graph did not look at all ominous. The spikes at 03:50 and 04:20 are normal for a Saturday morning – they result from an upstairs machine performing a remote disk-to-disk backup to the downstairs backup server.

Figure 3 is a more detailed view of the downstairs network, and shows that the backup server (called cow) receives backup data from upstairs, and additionally gets backup data from maxwell at 03:30 and samantha at 05:30. But more ominous (again, only in retrospect) is the outbound data from maxwell (which is also my mail server) starting at 06:00, and the same time, the relatively large amount of output from ns. Typically, the name server accounts for a negligible load, and is all but imperceptible in the graphs.

Indeed, except for data recognized in hindsight, there was nothing to indicate that anything was amiss. The network traffic was cursorily “normal” (although the traffic from the name server was “unusual”), and

³A myriad of email form scripts get this wrong by forbidding ‘+’ or other legal address characters. For a complete and correct regular expression for parsing email addresses, consult Jeffrey Friedl’s excellent book “Mastering Regular Expressions” (ISBN 0-596-00289-0), or look at <http://examples.oreilly.com/regex/email-opt.pl>

⁴I use Cricket (<http://cricket.sourceforge.net/>) to collect the data, RRDTool to store it, and drraw (<http://web.tar-nis.org/drraw/>) to display it in “dashboards.” Drraw is indispensable, because it allows unrelated data to be collected in a single view.

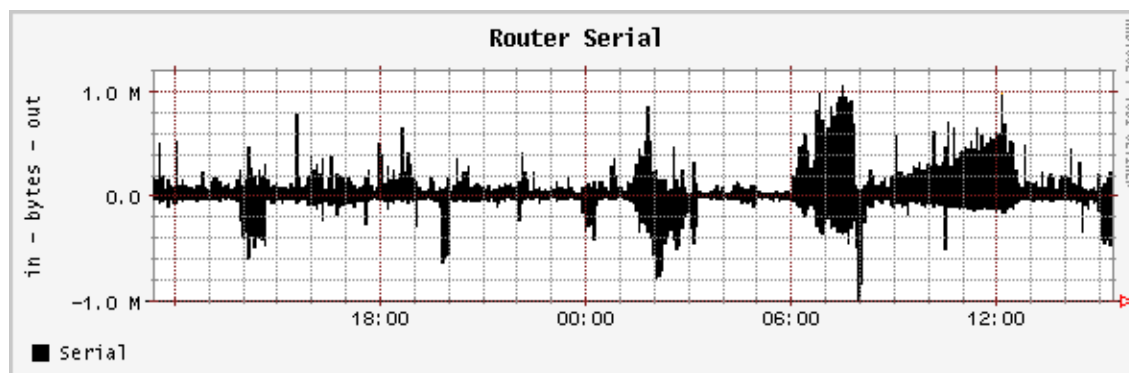


Figure 1: Overall network traffic.

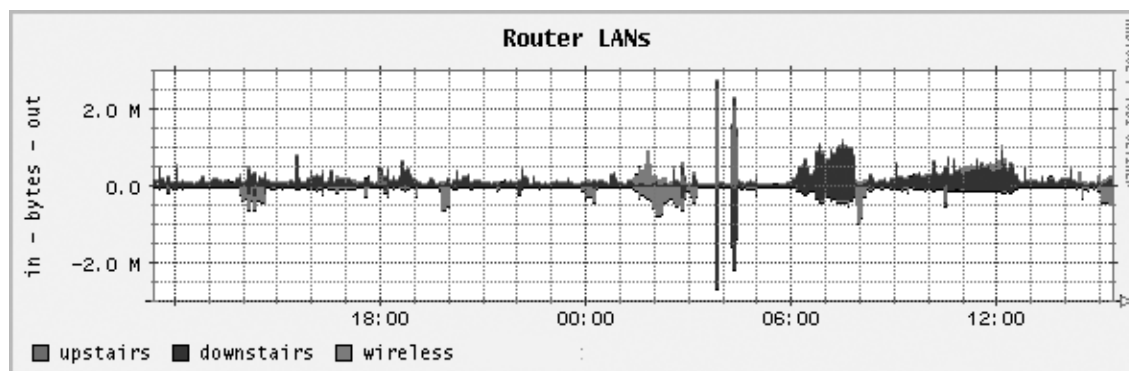


Figure 2: Internal network traffic by LAN.

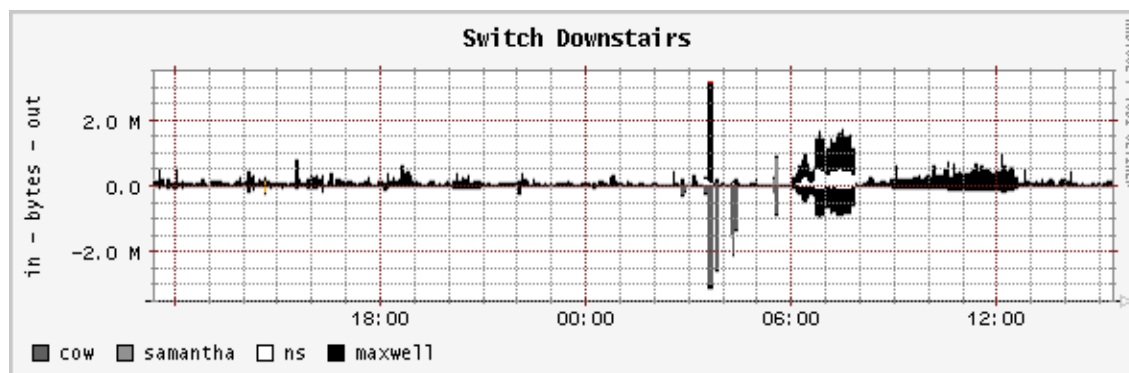


Figure 3: Internal network traffic on the downstairs LAN.

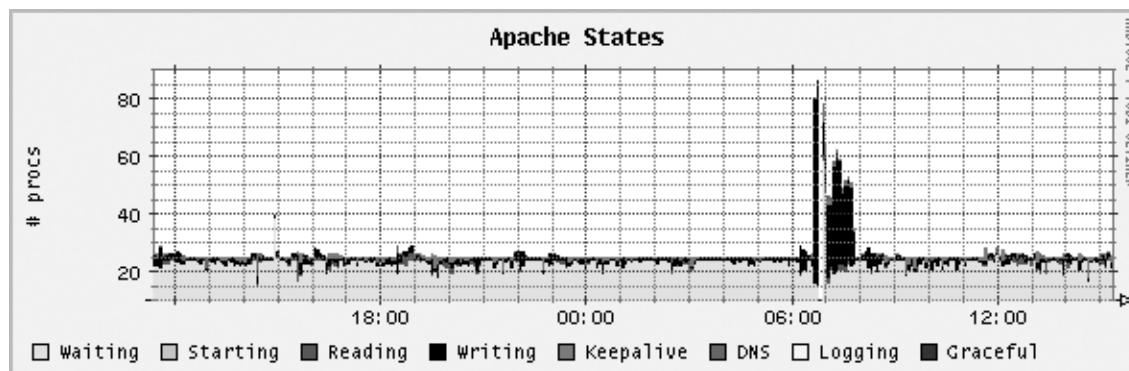


Figure 4: Apache States.

if network traffic had been my only critical indicator, a quick visual survey would have raised no red flags.

I had received a number of odd mail messages from the script that was being attacked, but I wrote those off as “someone messing with the site” (and was unconcerned given that I thought my scripts were secure). I also received one email message for each of the actual attack messages (since the script was *designed* to send me that mail). However, after the first few, my desktop spam filters “helpfully” blocked those messages, and except for the SNMP data, I was otherwise unaware of the thousands of spam messages being sent to AOL from my machine.

What ultimately triggered further analysis was the graph of Apache states, found in Figure 4 (however, this is only after first noticing the highly unusual

mail statistics in Figures 10 and 14, below). Mine is a lightly loaded web server, with MinSpareServers and MaxSpareServers set to 15 and 25, respectively. Typically fewer than a half-dozen servers are active serving requests. Thus when Apache reported 40 to 80 active processes for a span exceeding an hour, I determined that something was amiss.

It is especially noteworthy that there was no surge in the requests made of the webserver, nor in the amount of network traffic that the webserver was generating (seen in Figures 5 and 6). Thus each writing process (shown in Figure 4) was taking a lot of time to perform its job, but did so without generating a lot of data. Since Apache does not block on locked files, the only reasonable explanation was CGI script execution, where each script was taking a long time to complete.

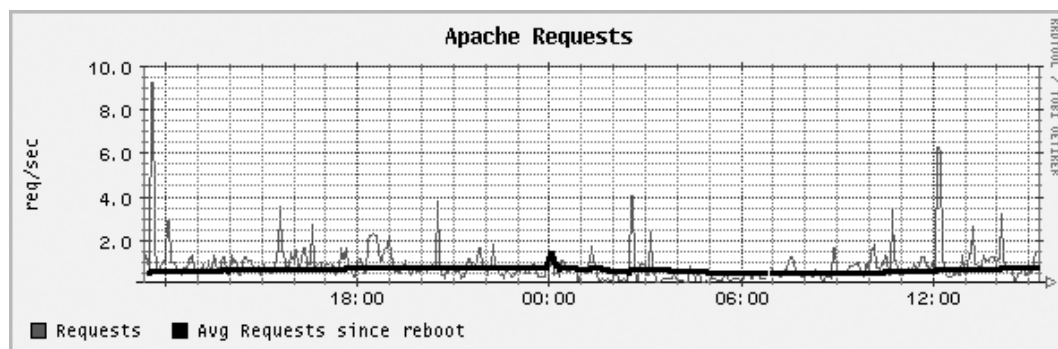


Figure 5: Apache requests from the web.

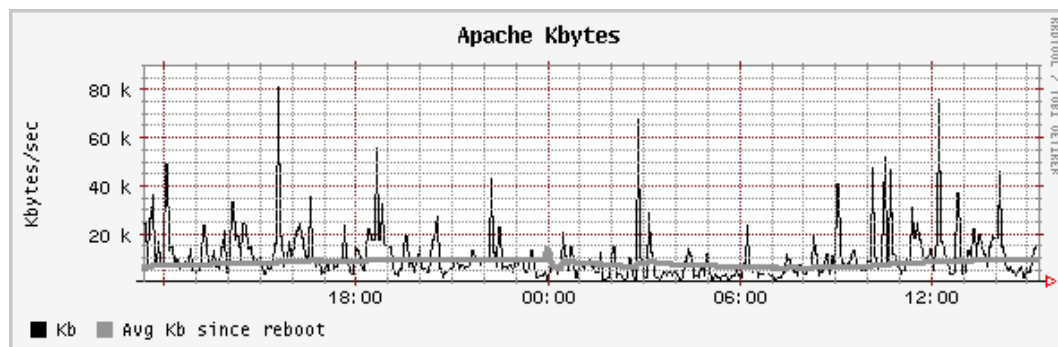


Figure 6: Kbytes served to the web by Apache web server.

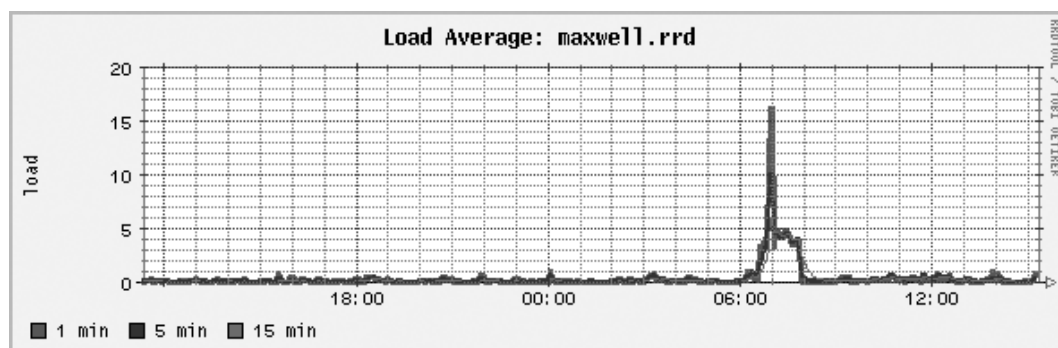


Figure 7: Load Average of Mail Server.

Indeed, as Figures 7 and 8 show, there were a large number of active processes. The load average on the machine (which is typically below 1.0) surged to a high of 16 before Sendmail's RefuseLA safeties cut in, and the number of active processes nearly quadrupled from 200 to 700.

Figure 9 shows the load on the name server machine. Typically, it putters along with a load average of 0.1 or less, but the large volume of email consumed nearly all of the resources of this machine. This small machine has a single purpose, and under ordinary conditions adequately services 15 internal hosts (all of whose web servers are configured to do reverse lookups), and is the authoritative name server for 185 domains. Therefore the load on the name server

presented by the spam attack is vastly in excess of any normal load experienced.

The volume of mail delivered (and deferred) by the attacked webserver is shown in Figure 10. From the start of the attack, the messages actually delivered (labeled "forwarded to a remote client") steadily climbs until it reaches a peak of nearly four messages per second. At approximately 08:00, AOL graylisted my server, the deliveries dropped, and the number of deferred deliveries grew steadily.

The peaks in deferred deliveries seen every 30 minutes are Sendmail re-running the queue (because the -q30m option was specified at startup). I estimate that between 15,000-20,000 pieces of spam mail were sent until AOL graylisted my server. As will be seen below,

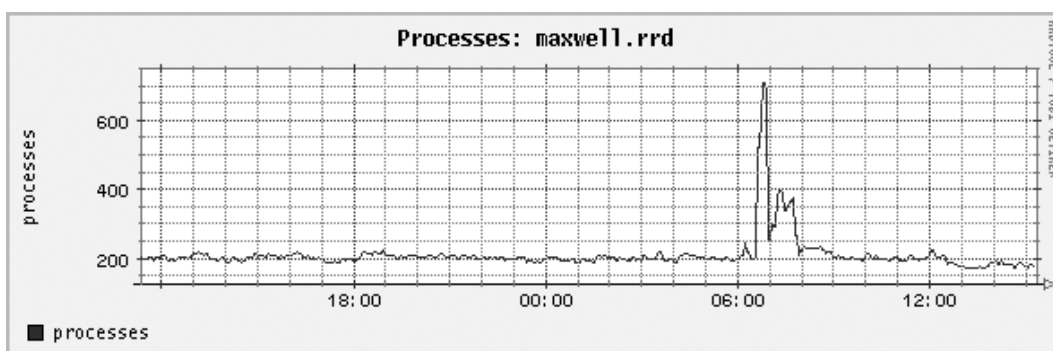


Figure 8: Number of Processes Active on Mail Server.

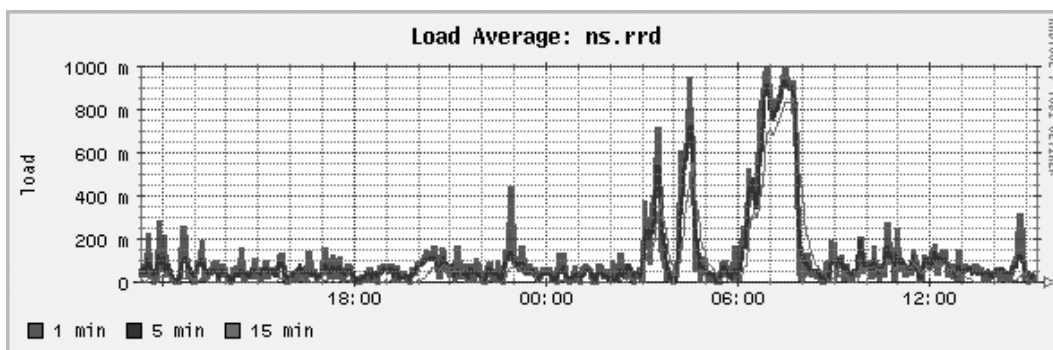


Figure 9: Load on Name Server.

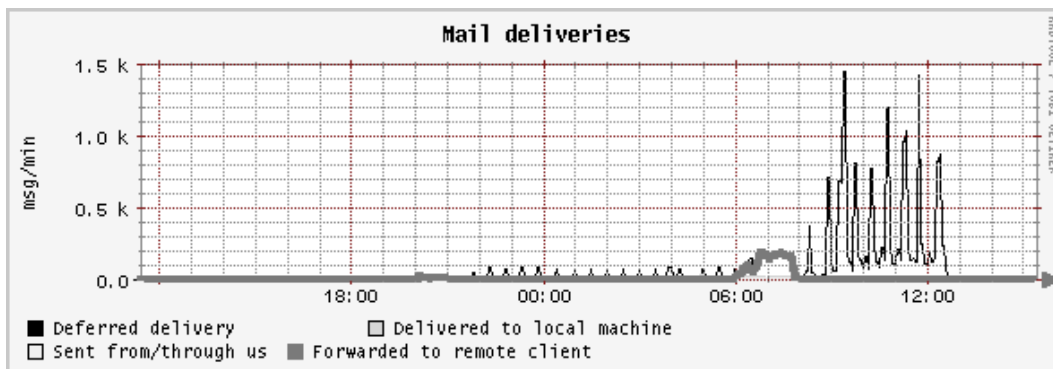


Figure 10: Mail Deliveries Made by Attacked Host.

this is a fraction of the potential impact on both my server and AOL. However, mine was but one of potentially hundreds of thousands of vulnerable email scripts.

Figures 11 and 12 show the number of message in the outbound mail queue, waiting to be sent. The number climbs steadily following the onset of the attack, and peaks at around 07:45, when the attack ceased (the precipitous drop off at 12:30 is due to my manually purging the queue). These numbers may appear small to someone who is used to running a larger site, but the sparseness of these numbers is misleading.

At its peak, the outbound mail queue contained 5,000 messages, with 80 MB of data. This may seem like a pittance, except when you consider that each message was typically addressed to 380-390 victims, so a queue size of 5,000 messages represents nearly

two million email targets. For a medium to large host, this additional traffic might go completely unnoticed, and I posit that the problem of unprotected scripts is far larger than we suspect!

It is essential to note that no one bit of data would have triggered my awareness of this spam attack (and the search that followed). It is only in aggregate that the data indicated something was amiss. In fact at some sites, an increase in queue size such as this might only suggest a stuck queue, and merely prompt a restart of the mail server. Likewise, if only the queue volume was used as a metric, this graph could easily have been confused as a user emailing a large file (such as a movie or collection of photos) that was not being accepted by the remote site. Plus, 80 MB of outbound mail is a pittance for many ISPs.

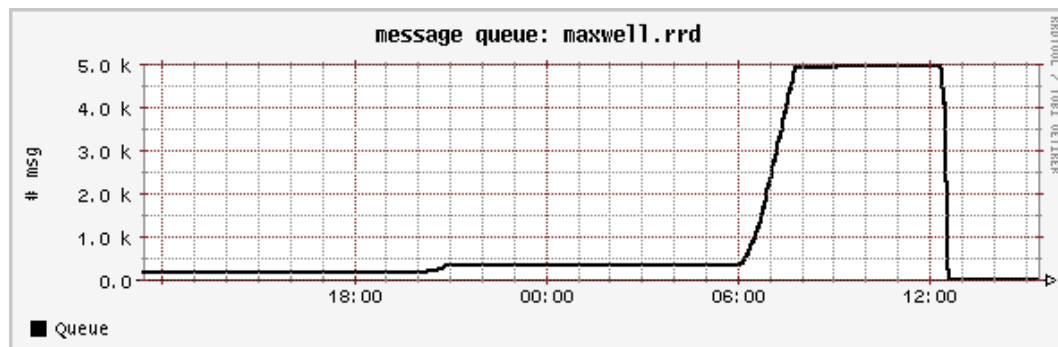


Figure 11: Number of messages in the outbound mail queue.

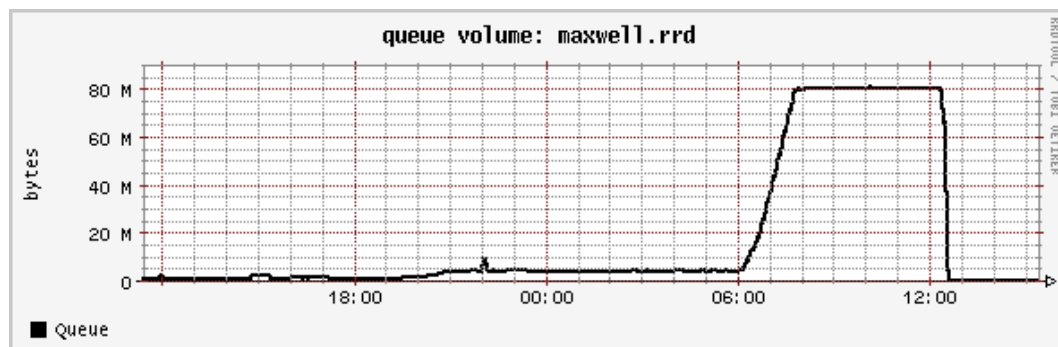


Figure 12: Volume of message in the outbound mail queue.

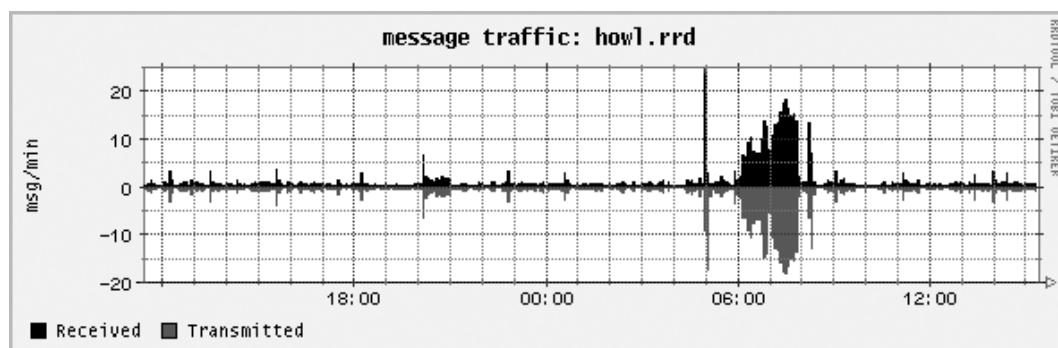


Figure 13: Mail traffic to my desktop.

Because my desktop spam filters deleted the large fraction of confirmation and spam bounce emails addressed to me, the volume of these messages was not immediately obvious. However, as Figure 13 shows, the number of email messages that were received (and automatically deleted) is greatly in excess of any “normal” traffic that my desktop would otherwise see.

Yet what is perhaps most interesting is the spam statistics for the machine that was attacked. As you can see in Figure 14 (which tracks types of spam *received*), there is nothing to indicate that spam is being *sent*. During the attack, the spam ratios and rates look completely normal (the spike of “unknown users” at noon is a different, but unrelated attack).

However, Figure 15 is very interesting. This graph tracks the ratio of spam to ham (and also tracks that which is unknown). Surprisingly, the amount of ham goes up during the attack. This is for two reasons. The first is that the exploited script will still send an email to the originally intended email address within our site. This is definitely considered ham, because it originates *inside* our network and is targeted to another machine *inside* our network. The second is because the bounced emails (that fail to reach their intended target at AOL) are unknown. They cannot be classified as either spam or ham, so again the ratio of spam to ham goes down. It was ultimately this graph that caused me to look more closely at what was

happening, since nothing (at the time) could immediately explain that erratic behavior.

Analysis of the Attack

The fact that I was attacked (and the solution to the vulnerability) was now obvious. The biggest problem I still had is that I didn’t know exactly *how* I was attacked. I know the reason (sending spam email), the target (a few million email addresses), the vector (my faulty CGI script), but not the source! In the 24-hour period with the most accesses of my vulnerable script, my web logs show a collection of 259 source-IP addresses scattered around the world.

Since web-servers report on the source-IP of the host that made the connection, my initial assumption was a collection of virus-infected Windows machines were attacking me, coordinated through some central database server. These compromised hosts could likely comprise a bot-net, but a problem exists with this hypothesis – a number of the machines were UNIX- or Linux-based machines! I attempted to contact a web server port 80 for each of the 259 machines. 27 of these identified themselves as UNIX/Linux Apache servers, 13 as Windows IIS servers, 6 as Squid proxies, and 6 as other web servers (the remaining 207 machines did not respond to queries on port 80).

I also considered that the attacker was using forged source IP addresses, but this possibility was quickly ruled out. Certainly, the SYN packets could

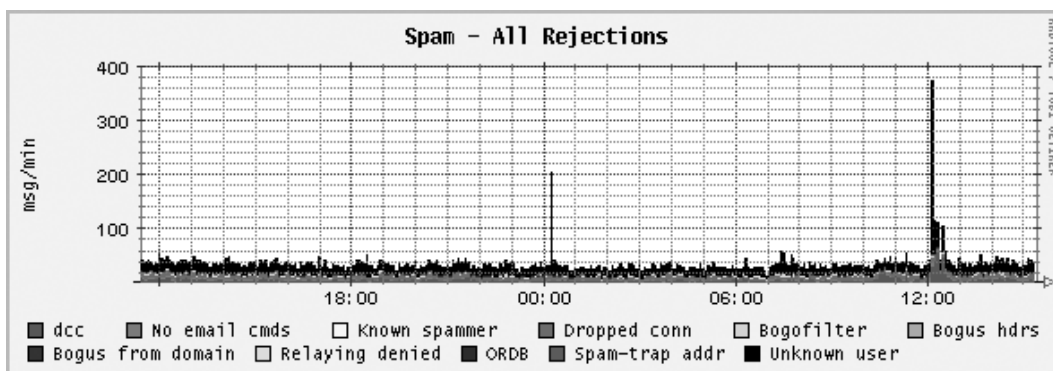


Figure 14: Spam detection.

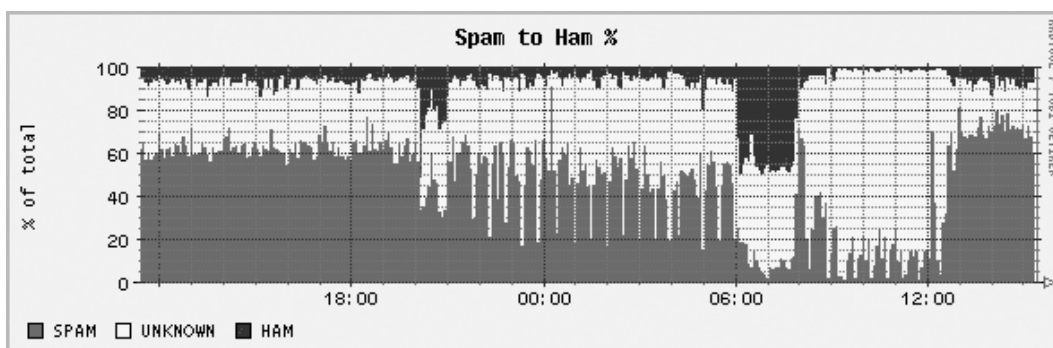


Figure 15: Spam to Ham Ratio.

have come from a forged address, but in order to make a request from my web server, the 3-way TCP handshake needs to be completed. The only way this could be done is for the attacker to be in the routing path of the forged address, and to make sure that the real machine does not respond to any response packets. While theoretically possible, this type of attack is highly unlikely, given the wide distribution of IP addresses.

My next thought was that these machines had open web proxies on them, and this seems the most likely explanation. Using *nmap*, I scanned each of the machines involved. Since my probes were done some months after the attack, it is very likely that some of the machines were either reprovisioned or reconfigured,⁵ and approximately 45% of the machines probed were reported as “down” (some of these responded very erratically, with long connection delays that often timed out). Of the remainder I determined that a large number (approximately 50%) of the machines were currently running web servers or proxy servers on “obvious” ports (80, 3128, 800x, or 808x). Some of the proxy servers currently required authentication, and although I did not have the proper credentials, it is highly likely that the spammer did. Sniffing software is readily available to find proxy passwords on unencrypted networks. There are also hundreds websites listing “free proxies,” and hundreds of others that share passwords – the combination of services is equally likely. I also have no doubt that some of the machines were running proxy servers on unobvious ports as a result of a virus or trojan with an HTTP proxy payload

Finally, I considered compromised logins, especially for those machines running UNIX or Linux. I found that approximately 25% of the machines had open *ssh* or *telnet* ports, so this hypothesis is possible, but it seems much more likely that open proxies are the vector for the attack.

In general, the attacker was smart. A small number of test emails were sent, and once a successful exploitation was found, a relatively small number of spam messages were sent (only a couple of thousand), and then the spammer moved on, presumably to another victim.

Because the attack was distributed through a collection of proxies, it is very difficult to determine the source of the attack. To do so would require accessing the proxy logs from a collection of the participating machines (which would require international cooperation from people who may not even know that they have proxy running), and this effort might in turn discover that the participating proxies were contacted by *other* proxy to further obscure the attack. And although this might eventually lead to a single source ISP, all that we would discover would be a temporary

⁵A webmaster that I contacted directly indicated that on the date of the attack he indeed had had a proxy running, but had since taken it offline due to “configuration issues.”

AOL account (to verify that the test emails were received), used only for a day or two and probably created using a forged or stolen credit card.

Further, although the spammer used my script and MTA to send approximately two million spam emails, the volume was not so large that my MTA was swamped, nor would the load have persisted for more than a few days. For a larger ISP, the excess web and email load might even go completely unnoticed!⁶

Once the script was used to send emails, renewed probes occurred daily for a few days as two to five tests were sent per host. Since I had by this time fixed the bug in the script, this probing gradually trailed off for the month after the initially discovered attack. However, it seems that the attackers are not fully coordinated, as my script is still probed a few times almost every day from locations around the world. I am “on their list,” and it seems that it will take quite some time before I am removed (if ever).

Lessons Learned

A number of lessons can be learned from this attack. Many of them are “obvious,” in that they are things that I should have learned and taken to heart ages ago. *Experience* is recognizing a mistake when you make it the second or third time, and I have plenty of that. *Security*, on the other hand, is not allowing oneself to make the same mistake the second time! I pass on my experience in the hope that my readers may have better security.

- Audit your scripts. It is far too easy to fall into the trap of thinking “if it is available on the web, it must be good.” The truth is of course “not by a long shot,” but internally developed scripts are equally vulnerable, and should also be audited. If you have a colleague who can audit your scripts, have them do so. But even a lay person can help you audit. I call this the “mannequin theory of programming.” If I take the time to explain my code to a total dummy, then I slow myself down enough to think about what might go wrong (and can then make notes to fix it).
- Notes are useless without a corresponding action. “I’ll get to it” has severe consequences when you *don’t* get to it! Once you find bugs, fix them!
- Finding a bug in one script probably means that the same bug exists in others. When you find a bug, exhaustively search your site(s) for scripts which have similar behavior to see if the mistake has been repeated.
- Audit your configurations. Most UNIX/Linux systems have Apache and Squid trivially available to them, and both can be (mis?)configured

⁶The open proxies are perhaps not as lucky. Some of the open proxies I examined had response delays of 45 seconds or more. While they may be intentionally throttled to prevent abuse, they are more likely seriously overloaded by that selfsame abuse.

as an open reverse proxy. While I will not engage in the privacy and anonymity arguments that proxy use can engender, I will only note that with every freedom there is a concomitant potential for abuse of that freedom. If you intentionally run a proxy, make sure it does what you intend it to do.

- Monitor your network. Daily dashboards are great for detecting anomalies, but beware of slowly changing things, too. A slowly filling disk (due to non-deletion of old log files) wasn't noticed for a year, because I only looked at daily, weekly, and monthly logs (and only the yearly log showed the slow trend)!
- Don't rely on automated detection. Tools like IDS systems are only as good as their processing rules (that in general rely on past experience), and they tend to discourage vigilance and encourage laziness. For example, would *cfengine* have found this attack? In a private communication, Mark Burgess (author and designer of *cfengine*) only says "maybe." More accurately, he said:

The *cfenvd* daemon would certainly have noticed the rise in SMTP, DNS, processes etc., but the question is – would your monitoring policy have bothered to report it to you? In *cfengine*, you have to tell it what you want to hear about. The default is always – nothing!

Cfengine does a kind of "lazy evaluation" on the data. If you ask to see anomalies in, say SMTP, you can also ask it look more closely at the distribution of IP sources. It uses the "informational entropy" (i.e., how sharp the distribution is) as a policy parameter. So you could say "tell me only about low entropy SMTP anomalies" (meaning tell me only about SMTP anomalies that come from one or proportionally few IP sources – a sharp attack from a single source). Or you could say "show me all of them," or "high entropy" (wide range of sources). So as with all systems, it depends on how you had it configured.

- Know what you are looking at (and looking for). If you don't know your data, then it is that much harder to recognize an anomaly. In my case, a single anomalous graph would not have triggered my search. Instead, it was the correlation of a number of unusual bits that did so. Your data should provide specific information as well as the overall gestalt of your network.
- Know your systems and your configuration, and where possible, leave yourself clues. Troubleshooting is as much an art as it is a science, and the more you know about your environment, the more likely you are to intuit the source

of a problem. In my case, I immediately recognized the 10-year old script I had written from the text of the email messages it was sending. However, even if I hadn't, I had unique text in every email (tagged to the script) that would have enabled me to quickly grep for the offending script. An even better clue would be to put both the URL and the location of the file on your server in the script output.

Author Biography

Daniel Klein has been teaching a wide variety of Unix-related subjects since 1984, and has been involved with Unix since 1976. His experience covers a broad range of disciplines, including the Internals of almost every Unix kernel released in the past 30 years, real-time process control, compilers and interpreters, medical diagnostic systems, system security and administration, web-related systems and servers, graphical user interface management systems, and a race-track betting system. He contributes regularly to the proceedings of the USENIX Association, and is also their tutorial coordinator. He holds a Masters of Applied Mathematics from Carnegie-Mellon University in Pittsburgh, and in his free time is director of an a cappella group and a member of an improvisational comedy troupe.

Conclusion

Spam is a problem that is going to plague us as long as we provide a means for spammers to send their messages. Crime may not pay, but spam certainly does (it has been estimated that Jeremy Jaynes was earning \$400,000 to \$750,000 *per month*). While providing an open mail relay or an open web proxy may be viewed as an exercise in free speech, CGI scripts which allow email to be sent to any address other than the intended one can only be viewed as software with a bug. Fixing this particular bug is relatively easy, and has an immediate beneficial effect. While the overall impact of fixing this vulnerability on the volume of spam can only be speculated at, it is unquestionable that the volume *will* be reduced by doing so.

Firewall Analysis with Policy-Based Host Classification

Robert Marmorstein and Phil Kearns – The College of William and Mary

ABSTRACT

For administrators of large systems, testing and debugging a firewall policy is a difficult process. The size and complexity of many firewall policies make manual inspection of the rule set tedious and error-prone. The complex interaction of conflicting rules can conceal serious errors that compromise the security of the network or interrupt the delivery of important services. Most existing tools for verifying the policy require the user to provide a detailed set of test cases or queries, which can sometimes be as difficult as verifying the policy by hand. Deriving a sufficiently comprehensive set of tests requires a detailed knowledge of potential vulnerabilities and a familiarity with the mechanics of the firewall. It also requires a significant investment of time and other resources. In this work, we present a fully automatic technique for identifying significant anomalies in a firewall policy. Our technique employs a novel system for classifying the hosts of a network into classes based on an equivalence structure, which is calculated from the firewall policy. This “policy-based classification of network hosts” substantially reduces the difficulty of identifying potential errors in the configuration of a firewall or group of connected firewalls and can be combined with existing firewall verification techniques to improve their effectiveness in detecting errors.

Introduction

A comprehensive firewall policy can protect a network against many serious internal and external threats. For installations with many hosts, however, the process of maintaining a restrictive policy can be prohibitively difficult. Firewall rules often interact in confusing ways. As the topology of the network evolves and users request new services, rules must be added and removed from the policy. These modifications make the policy increasingly complex and increase the likelihood of introducing serious errors into the firewall policy. For this reason, many system administrators avoid restrictive firewall policies and rely on other security measures to keep the network secure [17]. This is unfortunate, since a properly configured firewall can play a major role in defense-in-depth and in protecting legacy or unpatched systems.

One reason firewalls are so difficult to manage is that slight differences in the rule set can cause dramatic changes in the behavior of the firewall. For instance, on *iptables* firewalls [5], the filtering policy is specified using ordered chains of rules. In each chain, the first rule that matches a packet is used to determine the fate of the packet. Reversing two rules can introduce an error that is difficult to detect, but significantly modifies the behavior of the firewall. Other firewall systems have similarly deceptive semantics. For instance, *ipfilters* firewalls use a “last-match” policy to determine the fate of the packet [16]. Features such as network address translation and stateful filtering can also create opportunities for introducing difficult-to-detect errors.

While the techniques we describe in this work can be applied to any packet filtering system, we use *iptables* rules in our examples. An *iptables* firewall has three built-in chains. The INPUT chain is used to process packets that are destined for the firewall host itself. The OUTPUT chain is used for packets generated by the firewall host. The FORWARD chain manages packets that pass through the host to other machines. The policy language also provides the ability to define new chains which can be called from existing chains by making them the target of a firewall rule.

Each of the built-in chains consists of a default policy and an ordered list of rules. When deciding whether to accept or drop a packet, the firewall considers the rules of the appropriate chain in order until a matching rule is found.

Each firewall rule consists of a set of matches and a target. The matches specify criteria that determine which packets should be processed by the rule. The target tells the firewall what to do with a packet that matches. The target ACCEPT means to allow the packet through the firewall. The targets DROP and REJECT mean to discard the packet. The user can also instruct the firewall to pass the packet on to a user-defined chain for more processing or LOG the packet to a file.

Figure 1 gives an example of a simple *iptables* policy. The INPUT chain of the firewall allows SSH traffic to reach the firewall host itself. The default policy of DROP ensures that all other traffic to the host will be discarded. Outgoing traffic from the firewall, however, is allowed to pass. Although the OUTPUT chain contains no rules, the default policy of ACCEPT allows the firewall host to send any packet to any host.

The FORWARD chain of the firewall protects an internal network 192.168.1.0/24 against threats from the outside world. Only SSH and HTTP connections to internal machines are allowed. All other traffic is blocked. HTTP traffic will only be accepted if it is bound for the web server (host 192.168.1.3).

Even a simple policy such as this one can be difficult to maintain. Suppose, for instance, that a new wireless network is brought online as subnet 192.168.2.0/24. To better protect the existing internal network, the system administrator may add a rule to the firewall policy that blocks SSH traffic from the new network. Figure 2 shows one way the system administrator might modify the policy.

Rule 3 of the new FORWARD chain is intended to prevent hosts on the wireless network from connecting to protected machines. Unfortunately, the order of Rules 1 and 3 creates a hole through which SSH traffic can enter the protected network. As described above, the firewall processes the rules from

top to bottom and applies the first rule that matches. When an untrusted machine from 192.168.2.0/24 tries to access a protected machine on the 192.168.1.0/24 subnet, Rule 1 will grant access and Rule 3 will never even come into play. A correct policy, in which the rules are ordered properly is given in Figure 3.

In a simple three rule policy, errors like this can be easily detected and corrected. Many firewalls, however, have dozens or even hundreds of rules. Debugging these larger policies can be extremely difficult.

Existing Tools

The difficulty of testing the firewall can be reduced by using firewall analysis tools. Existing tools fall into three general categories: active testing tools, passive testing tools, and structure analysis tools. Active testing tools check for specific vulnerabilities by transmitting packets over the wire. Passive testing tools perform an off-line analysis to answer user queries about the policy. Structure analysis tools

INPUT (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	Anywhere	Anywhere	TCP dpt:22
OUTPUT (Default ACCEPT)				
#	Target	Source Address	Destination Address	Options
FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	Anywhere	192.168.1.0/24	TCP dpt:22
2	ACCEPT	Anywhere	192.168.1.3	TCP dpt:80

Figure 1: A simple *iptables* firewall.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	Anywhere	192.168.1.0/24	TCP dpt:22
2	ACCEPT	Anywhere	192.168.1.3	TCP dpt:80
3	DROP	192.168.2.0/24	192.168.1.0/24	TCP dpt:22

Figure 2: Adding a rule to the chain.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	DROP	192.168.2.0/24	192.168.1.0/24	TCP dpt:22
2	ACCEPT	Anywhere	192.168.1.0/24	TCP dpt:22
3	ACCEPT	Anywhere	192.168.1.3	TCP dpt:80

Figure 3: A correct policy.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	192.168.1.*	Anywhere	TCP dpt:80
2	ACCEPT	Anywhere	Anywhere	state ESTABLISHED

Figure 4: Forwarding chain of a stateful firewall.

identify poor practices such as duplicate rules in the specification of the firewall policy.

Active Testing Tools

Port scanners such as *nmap* [7] and *hping* [3] can be used to simultaneously test the firewall and important servers. Vulnerability testing tools [6, 10, 9, 2] also allow the system administrator to detect common firewall errors. These tools are “active tools” in the sense that they test the security of the network by transmitting packets through the firewall. Although this has the advantage of testing both firewall and host, it has the disadvantage of only detecting errors when the host is available. If a host is down for maintenance or powered off by a user, these tests cannot be used to verify the firewall policy. Furthermore, because active tools consume a significant amount of bandwidth and CPU time, they can only examine a small subset of the possible packets a firewall might encounter. To thoroughly test every possible combination of source address, destination address, and network port is infeasible.

Active tools require the user to decide which behaviors to test. For instance, a port scanner requires a list of hosts to scan. Scanning all ports on a few important servers will often catch the most critical vulnerabilities, but it is often helpful to also scan individual workstations for less obvious errors. To check for as many vulnerabilities as possible, the user must carefully craft a testing pattern that balances running time against the number of hosts to scan, the number of ports to check, and the number of spoofed source addresses to employ. Vulnerability scanners such as *Nessus* [10] also require a significant amount of user input. These tools make use of a database of pre-designed tests. While well-known vulnerabilities can usually be caught using the scripts provided with the scanner, creating new tests requires learning a sophisticated scripting language.

Passive Testing Tools

One alternative to active testing is off-line analysis of the firewall policy using a query engine. Passive tools [15, 18, 11, 4] use efficient data structures to analyze a representation of the firewall policy without transmitting any packets over the network.

Passive tools can be much faster than active tools and do not interfere with other network activities. In theory, passive tools can test every possible behavior of the firewall. In practice, however, such a test produces too much unstructured output to be useful. Since the decision of which behaviors are desirable and

which are undesirable must be made by the user, testing all eventualities would produce output for every possible packet seen by the network. Since there are 255^4 possible source addresses and the same number of destination addresses, there are billions of packets to consider – an overwhelming amount of output.

To avoid this problem, the user must carefully construct a set of queries that test for specific vulnerabilities. While it is often easier to construct these tests than to inspect the rule set manually, it can be difficult to create queries that test enough interesting behaviors and produce useful output. As with active testing tools, there is no way to guarantee that all important behaviors have been tested. If the system administrator fails to provide a test for an important threat, the testing software cannot detect that the firewall is vulnerable. Errors that are difficult to catch by manually inspecting the rule set are also likely to be overlooked by the user when creating queries for a passive tool and escape detection.

Subtleties in the syntax of the query language can also cause the query engine to generate unexpected results. In previous work, we introduced *ITVal* [12, 14, 13], a passive analysis tool for *iptables* firewalls. Our tool uses decision diagrams to store a representation of the rule set and allows the user to perform queries such as “From which hosts is SMTP allowed to the mail server?”

```
QUERY DADDY FOR TCP 80 AND
  NOT FROM 192.168.1.*;
# Addresses: *.*.*.*
```

Figure 5: An example *ITVal* query.

The *ITVal* query given Figure 5 might be used to discover which servers provide web access to hosts outside the network. The “DADDY” subject tells *ITVal* to list the destination addresses of these machines. The query condition “FOR TCP 80” specifies a match against all HTTP packets while the condition “NOT FROM 192.168.1.*” excludes internal hosts from consideration. (For a more detailed treatment of the syntax of *ITVal* queries, we refer the user to [14].) For many firewalls, the query in Figure 5 will work as expected. However, for a stateful firewall, such as the *iptables* rule set of Figure 4, it is likely that this query will generate many false positives.

When *ITVal* processes the query against the stateful firewall, it will report that any host can send web traffic through the firewall. This surprising result is technically true. The rule on line 2 of Figure 4 allows

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	10.239.202.38	Anywhere	dpt tcp:25
2	ACCEPT	10.239.202.0/24	10.239.202.38	dpt tcp:25

Figure 6: Controlling mail with a packet filter.

arbitrary access on established connections. A more careful examination of the rule set, however, reveals that only machines on the internal network can initiate new connections to the web server. A more precise query that examines only new connections is given in Figure 7. The new query correctly reports that only internal hosts can initiate HTTP connections.

```
QUERY DADDY FOR TCP 80 AND
NOT FROM 192.168.1.*
AND IN NEW;
# Addresses: 192.168.1.*
```

Figure 7: A better query for stateful firewalls.

Structure Analysis Tools

Although query-based testing tools can be a significant help to the system administrator, they are limited by the user's ability to construct a comprehensive set of useful queries. It is difficult to tell whether a set of queries tests every important behavior of the firewall. Furthermore, testing techniques often generate too much output for the user to easily distinguish dangerous vulnerabilities from desired behavior.

Another approach to firewall analysis is to look for errors in the structure of the policy specification. Structure analysis tools [1, 8] detect problems such as duplicate or conflicting rules. Although these tools do not directly identify vulnerabilities, they often uncover fundamental weaknesses in the policy that can produce more significant errors. Some of these tools also generate a simpler version of the policy that removes these structural weaknesses. The generated policy is often easier to inspect manually than the original policy.

One significant advantage of structure analysis tools is that they can be fully automatic. The only input the user must provide is the firewall policy itself. The tool builds a list of anomalies and outputs a report or a restructured version of the policy. Unfortunately, there are many types of vulnerabilities that cannot be detected using these tools. For instance, allowing mail traffic from the outside world to certain workstations could be undesirable behavior on some networks. A structure analysis tool would not detect a problem of that nature unless the rule that permitted the flow of such traffic also conflicted with another rule or violated the structural criteria in some other way.

Host Classification

The "Lumeta Firewall Analyzer" [18], a commercially available tool derived from FANG [15], combines some of the advantages of a structure analysis tool with the flexibility of a passive analysis tool.

Lumeta automatically generates a comprehensive set of queries by using routing information to classify hosts into groups [18]. This reduces the amount of output since results can be provided on a per-subnet or per-zone basis rather than a per-host basis. It also removes the burden of designing good queries from the user.

The idea of classifying hosts into groups allows a query engine to provide much simpler output and addresses the problem of creating good queries. Using the topology of the network to classify hosts, however, has the drawback that hosts with very different properties, but that have similar addresses, are grouped together.

Consider, for instance, the filtering policy shown in Figure 6. This simple policy restricts outgoing mail from an internal network 10.239.202.0/24. Outgoing traffic is only allowed from the mail server, host 10.239.202.38. Other hosts on the subnet are allowed to send mail to the mail server, but cannot send mail to each other or to the outside world. Incoming SMTP mail traffic from the outside world is also dropped unless it is destined for the mail server.

A classification based on the network topology would break the network into two groups: the set of hosts on 10.239.202.0/24 and the set of all other hosts. However, the mail server is a different type of host from the other machines on the network. As a result, queries about mail traffic will return imprecise results. For instance, the answer to the query "Can a host in 10.239.202.0/24 send mail to the outside world?" will be "yes" since the mail server is allowed to forward mail through the firewall. The query "Can all hosts in 10.239.202.0/24 send mail to the outside world?" will be "no" since the client machines are not allowed to send mail to anywhere but the mail server.

Neither of these queries accurately describes the fundamental organization of the network: a special mail server which can send mail to the outside world and a set of clients which cannot. To improve the precision of these queries, we must use a different classification scheme that allows us to group hosts by their function as well as by their placement in the network topology.

Policy-Based Host Classification

Hosts on a network play a variety of roles. Some hosts are workstations. Some are database servers. Some are web servers. Some provide multiple services. Each type of host is often treated very differently by the firewall. Hosts of the same type, however, are usually treated very similarly. This means that the firewall implicitly classifies hosts into various groups

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	DROP	192.168.2.0/24	192.168.1.0/24	
2	ACCEPT	Anywhere	192.168.1.1	dpt tcp:80

Figure 8: A simple network with four host classes.

based on their function. Sometimes the implicit classification of the firewall policy is not quite as straightforward as simply sorting hosts by the services they provide. For instance, the network may have a web server that provides service exclusively to hosts inside the network as well as a general purpose web server that anyone can access. The filtering policy for these two systems could be drastically different even though they are both web servers.

The rule set in Figure 8 prevents hosts on an untrusted network 192.168.2.0/24 from accessing systems on a protected network 192.168.1.0/24. Rule 1 divides the set of hosts into three groups. One group consists of hosts in the untrusted network. The second group contains hosts from the protected network. The third group contains all other hosts on the Internet. Rule 2 refines this classification by further restricting which services are available to the web server. This distinguishes the web server from other trusted machines, creating a fourth group containing only the web server.

This classification scheme has many advantages over a topological classification. An error in the firewall policy will often cause the firewall to treat similar hosts differently or to treat different hosts alike. This means that a classification scheme based on the structure of the firewall policy can be used to directly detect many kinds of errors. Furthermore, classifying hosts according to their treatment by the firewall produces groups of hosts that can be used to increase the precision of query-based testing techniques.

Calculating Host Classes

There are several possible ways to use the structure of the firewall policy to create classes of hosts. A naive approach is to search through the firewall policy and record every group of addresses that is explicitly mentioned as a host class.

Unfortunately, the naive approach will generate overlapping classes. For instance, the host 192.168.1.1 from Figure 8 would be represented twice: once in its own class and once in the class containing all hosts from the 192.168.1.0/24 subnet. This is undesirable because it decreases the precision we can obtain in our queries. A host that appears in two classes is fundamentally different from the other hosts in those classes and should be categorized separately in order to obtain accurate results.

The algorithm in Figure 9 reduces the amount of overlap by splitting overlapping classes into smaller

pieces. The algorithm examines every host and set of addresses mentioned explicitly in the rule set. Each new range of addresses is added to a set of potential classes, C .

```

set CalculateClasses(Policy P):
1 set C = {0.0.0.0/0}
2 for each rule r in P:
3   for each addr_range S in r:
4     C = InsertAddr(C, S)
5 return C

set InsertAddr(set C, addr_range S):
1 for each element T of C:
2   I = IntersectAddress(S, T)
3   if I is empty:
4     C = SetAdd(C, S)
5 return C
6 C = SetDelete(C, T)
7 C = InsertAddr(C, S-I)
8 C = InsertAddr(C, I)
9 C = InsertAddr(C, T-I)
10 return C

```

Figure 9: A naive algorithm for computing host classes.

If a new set of addresses overlaps with an existing class, we break both classes into three non-overlapping pieces and replace both original classes with the result. When we have considered every address of every rule, the elements of C describe a set of classes that can be used to analyze the behavior of the firewall.

This approach yields an approximation of the firewall designer's view of the network. Addresses that are explicitly mentioned usually correspond to important components that the designer intended to control. Unfortunately, the technique does not give a perfect picture of the actual behavior of the firewall. For instance, the firewall rule set in Figure 10 seems at first glance to have three groups. The algorithm will create a group for subnet 192.168.2.0/24 and for subnet 192.168.3.0/24. It will also create a group representing "all other addresses."

In reality, hosts on the 192.168.3.0/24 subnet are treated no differently from hosts in the "all other addresses" group, because Rule 3 of the firewall policy is an unreachable rule. Because all packets from 192.168.2.0/24 will be dropped in Rule 1, no packet can ever match Rule 3. This is probably an error in the firewall configuration, but the naive algorithm will happily report that, as expected, 192.168.3.0/24 is a special class and the user will not detect the error.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	DROP	192.168.2.0/24	Anywhere	
2	ACCEPT	Anywhere	192.168.2.0/24	
3	ACCEPT	192.168.2.0/24	192.168.3.0/24	

Figure 10: Rule set with a shadowed network.

To correct this problem we need to more carefully define the concept of a host class. We do this by constructing an equivalence relation over the set of all network hosts. The equivalence classes determined by this relation will give us a precise and complete characterization of the policy that we can use for performing vulnerability analysis.

Structure-Based Classification

Every firewall policy can be described as a function, F , that maps the set of all network packets to the set $\{ACCEPT, DROP\}$ of filtering decisions. For a specific packet p , we say $F(p) = ACCEPT$ if the packet would be accepted by the firewall and $F(p) = DROP$ if the packet would be dropped by the firewall.

We define an equivalence relation, \equiv_{SD} , as follows: let x and y be any two hosts. We say that $x \equiv_D y$ if and only if for any two packets p from x and q from y that differ only by source address, $F(p) = F(q)$. Similarly, $x \equiv_D y$ if and only if $F(p) = F(q)$ for any two packets p from x and q from y that differ only by destination address. If $x \equiv_S y$ and $x \equiv_D y$, then we say that $x \equiv_{SD} y$.

Informally, two hosts are source equivalent if replacing the source address of a packet from one host with the source address of the other does not affect the filtering decision of the firewall. They are destination equivalent if replacing the destination address does not affect the filtering decision. If they are both source and destination equivalent, we say that they are equivalent under the relation \equiv_{SD} . The relation \equiv_{SD} is derived directly from the function F , which describes the filtering policy of the firewall and can be computed without any other input from the user.

It can be shown that \equiv_{SD} is an equivalence relation, since it is reflexive, transitive, and symmetric. This means that \equiv_{SD} partitions the set of network hosts into equivalence classes. In other words, a packet from a host in a particular equivalence class will only be accepted if identical packets from other hosts in the class would also be accepted.

This means that if one host in the class has a vulnerability, all hosts in the class are vulnerable. On the other hand, if that host is adequately protected by the firewall, then all the others are too. This guarantee makes the equivalence class paradigm much more useful than the naive classification algorithm or a classification based on topology for generating precise queries.

Implementation

In previous work [14], we described an algorithm for representing the rule set of an *iptables* firewall as a multi-way decision diagram (MDD). We can use the reduction properties of the MDD to compute the equivalence classes of the firewall.

An MDD is a directed acyclic graph that can efficiently represent a function over a large set of vectors. An MDD representation of the rule set of a firewall can be obtained by converting the rule set into a function over the set of all network packets. An

example MDD is depicted in Figure 11. Each of the non-terminal levels of the MDD corresponds to one field of a packet. For space and simplicity, we have greatly simplified the example by considering only a few of the fields. In practice, the MDD is much larger and contains levels for such values as the protocol, the source port, the destination port, and the TCP flags.

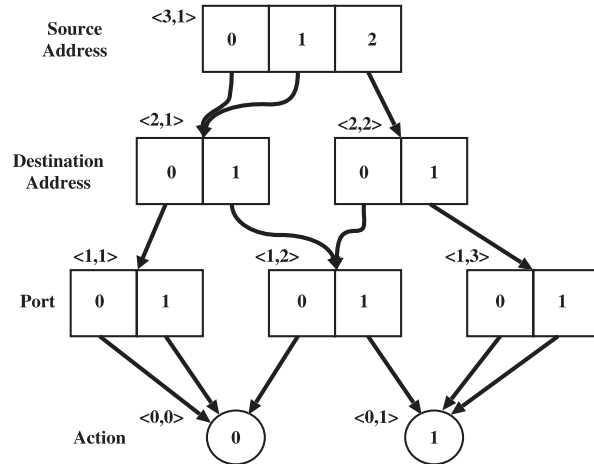


Figure 11: A simplified rule set MDD.

The top level of the MDD in Figure 11 corresponds to the source address of a packet, while the second level corresponds to the destination address of the packet. The bottom level is a special terminal level representing the action that the firewall should take on a packet. The integer value 0 means to drop the packet. The integer value 1 means to accept the packet.

Each path through the MDD represents one filtering rule. Each node represents a set of related packets. We use the notation $\langle k, i \rangle$ to denote the i^{th} node of level k . Each arc at level k represents a choice of value for attribute k of the packet. For example, the first arc of node $\langle 3, 1 \rangle$ represents a source address value of 0 and node $\langle 2, 1 \rangle$ of Figure 11 represents the set of all packets with source address 0 or 1. The path $\langle 3, 1 \rangle$ to $\langle 2, 2 \rangle$ to $\langle 1, 2 \rangle$ to $\langle 0, 1 \rangle$ represents the rule that any packet with source address 2, destination address 0, and destination port 1 will be accepted by the firewall.

1. Construct the MDD representation of each firewall chain.
2. For each chain:
3. Reorder the levels of the chain MDD so that source address is on top.
4. Record the source equivalence classes.
5. Reorder the levels of the chain MDD so that destination address is on top.
6. Record the destination equivalence classes.
7. Merge the source and destination classes of all three chains together.

Figure 12: Outline of the equivalence class computation algorithm.

In ITVal, we use reduced-ordered MDDs in which duplicate nodes, with all arcs the same, are not allowed. This requirement means that each node at level k represents an equivalence class over the set of attributes K through $k + 1$, where level K is the top level of the MDD. For instance, node $\langle 2, 1 \rangle$ represents the source equivalence class containing addresses 0 and 1. Node $\langle 2, 2 \rangle$ represents the class containing source address 2. By reordering the levels of the MDD, we can calculate equivalence classes over first the source address and then the destination address. We can use these intermediate classes to construct classes of hosts that are equivalent under the \equiv_{SD} relation. An outline of the class generation procedure is given in Figure 12. A more detailed illustration of the algorithm is given in Figure 13.

In step 1, we generate an MDD representation for each of the three built-in chains. The MDD representation takes into consideration network address translation and other packet mangling rules. We then consider each chain in turn. In steps 3 and 4, we compute a list of source equivalent addresses. To do this, we first use a level swapping algorithm to bring the levels encoding source address to the top of the graph. The reduction properties of the MDD now guarantee

that each node at the level immediately below the source address levels represents an equivalence class with respect to source address. Each path from the root node to a node at that level represents one element of the equivalence class associated with that node. Step 4 extracts these equivalence classes and stores them in a new MDD.

In steps 5 and 6 we perform an identical operation to collect a list of equivalence classes with respect to destination address. When we have considered source and destination address in every chain, we now merge the various classes together using MDD union, intersection and difference operators. Finally, we print the result.

Error Detection

The information provided by the host classification algorithm can be extremely useful for detecting errors in the firewall policy. The list of classes is usually much shorter and simpler than the rule set, so it is easier for a system administrator to examine.

Detecting Remotely Accessible Services

Simple errors such as typos and transpositions can often be detected by the presence of a strange and unexpected class of hosts. The policy in Figure 14 is

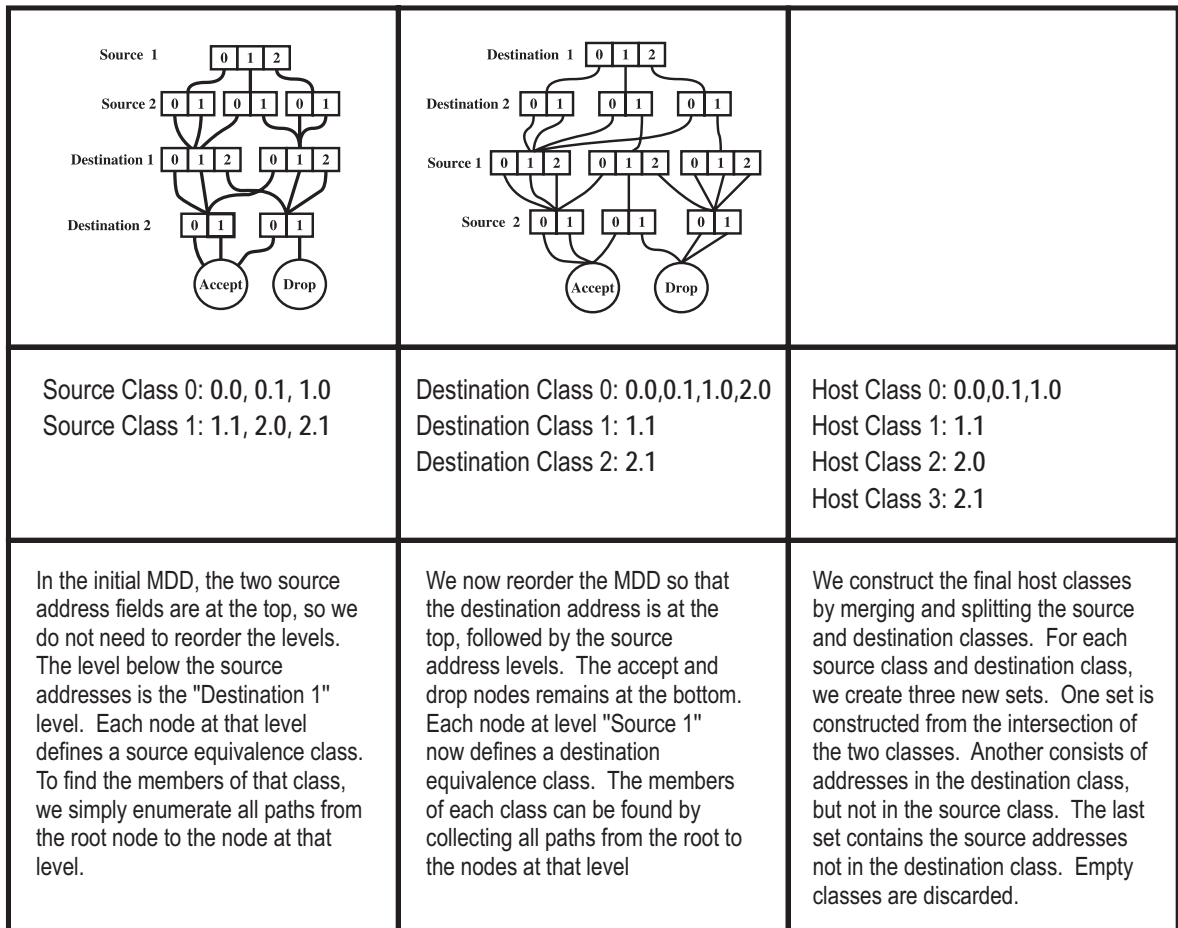


Figure 13: Step by step construction of the equivalence classes.

intended to protect networks 192.168.1.0/24, 192.168.2.0/24, and 192.168.3.0/24 by restricting access from the outside world. Because 192.168.1.0/24 contains several hosts with important financial information, outgoing traffic from that subnet should also be restricted. Mail traffic from the other subnets is allowed only to the mail server (host 192.168.2.20) to prevent compromised machines from becoming spam relays. The rule set also allows arbitrary web access to a group of web application servers located on the 192.168.3.0/24 subnet. The policy contains several errors, including a typo in Rule 3 that allows remote access to a protected service.

The equivalence classes of this example network are listed in Figure 16. There are five classes of hosts identified by the algorithm. Class 0 represents the group of web servers. Class 1 represents a strange class of hosts that exists because of the typo in rule 3. The strange, unexpected class makes the effect of the typo immediately obvious to the administrator.

Class 0: 192.168.3.*
 Class 1: 168.192.1.*
 Class 2: 192.168.1.*
 192.168.2.[0-19]
 192.168.2.[21-255]
 Class 3: 192.168.2.20
 Class 4: [0.0.0.0]-[168.192.0.255]
 [168.192.2.0]-[192.168.0.255]
 [192.168.4.0]-[255.255.255.255]

Figure 16: Equivalence classes for Figure 14.

Class 2 combines the protected financial network and the unprotected 192.168.2.0 network, minus the mail server. This should also arouse the analyst's suspicion since the financial network is supposed to have much stricter protection than the unprotected subnet. The fact that they are treated the same by the firewall indicates that a serious vulnerability exists. Class 3 contains the mail server. It is in a class by itself since

it requires special privileges in order to accept and relay mail. Everything else belongs to Class 4.

Using the equivalence classes to detect these errors is much easier than using query based tools. The presence of a class of hosts consisting entirely of strange addresses is a clear indication of an error in the policy. Since the tool requires no input but the policy, all the user has to do to discover the error is "fire and forget."

A small amount of work is required to interpret the results of the classification system, but compared to the effort of constructing precise queries or compiling a list of hosts for active testing, using the equivalence classes is really fairly simple. For large installations, the gain is even greater due to the number of rules required to administer a large number of hosts and the greater difficulty of specifying a comprehensive set of queries that covers all the services provided by the network.

Detecting Shadowed Rules

If a packet matches more than one rule in the policy, the firewall will use the first rule that matches. This can mean that the policy contains useless or unreachable rules. The presence of these rules usually indicates an error in the policy.

When one rule shadows another, the class list will often contain fewer classes than expected. For instance, the rule set in Figure 15 contains two rules that are shadowed by Rule 1. Rule 2 is a useless rule. Web packets from 192.168.2.0/24 to 192.168.3.0/24 are already accepted by Rule 1. Rule 3 is also unreachable. The class list for the example network is given in Figure 19.

Class 0: 192.168.2.*
 Class 1: [0.0.0.0]-[192.168.1.255]
 [192.168.3.0]-[255.255.255.255]

Figure 19: Equivalence classes for Figure 15.

Notice that there are no classes for the networks 192.168.3.0/24 and 192.168.4.0/24 mentioned in rules

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	Anywhere	192.168.3.0/24	TCP dpt:80
2	ACCEPT	192.168.2.0/24	Anywhere	
3	DROP	168.192.1.0/24	Anywhere	
4	ACCEPT	Anywhere	192.168.2.20	TCP dpt:25
5	ACCEPT	192.168.1.0/24	Anywhere	

Figure 14: Rule set with errors.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	192.168.2.0/24	Anywhere	TCP dpt:80
2	ACCEPT	192.168.2.0/24	192.168.3.0/24	TCP dpt:80
3	DROP	192.168.2.0/24	192.168.4.0/24	TCP dpt:80

Figure 15: Rule set with shadowed rules.

2 and 3. When the system administrator discovers that the policy produces fewer classes than expected, she will examine the policy more closely and detect the error. Shadowed rules often indicate that a rule contains an incorrect address. For instance, rules 2 and 3 may have become shadowed by a typo in Rule 1. This could happen if the source address in Rule 1 was supposed to be 192.168.3.0/24, but was typed incorrectly.

Detecting Outdated Services

Host classification can solve real world problems. One of our firewalls originally supported a wireless network on subnet 192.168.4.0/24. When wireless service was transferred to another network, we neglected to update the firewall rules. A portion of our rule set looked something like Figure 17. A quick analysis using host classification immediately identified subnet 192.168.4.0 as a host group, enabling us to correct the problem. This error would have been very difficult to detect using query-based analysis tools. Without apriori knowledge of the error, we had no reason to create a query testing for service on that subnet. Active analysis tools like Nessus would have detected no vulnerabilities, since no hosts were available on that subnet. Using host classification, however, we were able to immediately identify a serious weakness in our policy.

Using Equivalence Classes with Other Tools

While a system administrator can detect many important vulnerabilities simply by studying the host equivalence classes of a firewall policy, even greater gains can be achieved by combining the equivalence class analysis with active and passive testing techniques. To combine the analysis with other testing paradigms, we can use the equivalence classes to determine which systems to test. By taking one or two systems from each equivalence class, we can increase the probability that we have tested all the important behaviors of the firewall.

The filtering policy in Figure 18 secures the mail service on an internal network 192.168.2.0/24. Mail from the internal network can only be sent to the mail server, host 10.239.202.13. The mail server is allowed to distribute mail to both internal and external hosts. All other mail traffic should be dropped. Unfortunately, a copy and paste error created rule 3 of the policy, which allows mail traffic from a workstation, host 192.168.2.3 to escape the network. If that workstation is compromised, an intruder can set up a spam relay on that host and transmit thousands of unauthorized messages through the firewall.

Class 0: 10.239.202.13
 Class 1: 192.168.2.3
 Class 2: 192.168.2.[0-2]
 192.168.2.[4-255]
 Class 3: [0.0.0.0]-[10.239.202.13]
 [10.239.202.14-192.168.1.255]
 [192.168.3.0-255.255.255.255]

Figure 20: Equivalence classes for Figure 18.

The system administrator can easily detect this problem by combining host classification with a passive testing tool. The host classes for the example network are listed in Figure 20. By taking a source address from each of these groups and matching it with a destination address from each of the groups, we can construct the set of 16 IPv4 queries listed in Figure 21.

Notice that while the list of classes does not immediately cause concern, the query results show that 192.168.2.3 can access port 25 on host 192.168.3.0, which is in the “outside world” class of hosts. This strategy can identify serious problems without producing an overwhelming amount of input. Using the query tools by themselves would either have produced an enormous amount of data or required a large time investment in writing queries. Combining classification with passive testing limits the scope of

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	192.168.2.0/24	Anywhere	TCP dpt:22
2	ACCEPT	192.168.4.0/24	Anywhere	TCP dpt:8080
3	DROP	192.168.4.0/24	192.168.2.0/24	TCP dpt:25

Figure 17: Rule set with outdated rules.

FORWARD (Default DROP)				
#	Target	Source Address	Destination Address	Options
1	ACCEPT	Anywhere	10.239.202.13	TCP dpt:25
2	ACCEPT	10.239.202.13	Anywhere	TCP dpt:25
3	ACCEPT	192.168.2.3	Anywhere	TCP dpt:25
4	ACCEPT	192.168.2.3	Anywhere	TCP dpt:22
5	DROP	192.168.2.0/24	Anywhere	TCP dpt:25

Figure 18: Rule set for preventing spam relays.

the query to the important distinctions between hosts and requires very little work from the user.

Author Biographies

Robert Marmorstein plans to graduate with a Ph.D. from the College of William and Mary in the summer of 2007. When he is not actively researching ways to manage and analyze firewalls, he spends his time avoiding grues in the Great Underground Empire and tinkering with his collection of UNIX-based systems.

Phil Kearns is an Associate Professor of Computer Science at the College of William and Mary. His research interests lie in the general area of computer systems.

Conclusion

Policy-based host classification has several significant advantages over existing firewall analysis techniques. Examining the classes implicitly defined by the firewall policy allows a system administrator to detect many kinds of firewall errors and anomalies. When combined with active or passive testing tools, the technique can be even more powerful. Using the equivalence classes significantly decreases the amount of the work required to verify the policy and is a step toward a fully automatic firewall analysis solution. The latest version of ITVal, provides a special “CLASSES” query that displays a list of the host equivalence classes. The tool and some examples of its use can be found at <http://itval.sourceforge.net>.

References

- [1] Al-Shaer, Ehab S. and Hazem H. Hamed, “Modeling and management of firewall policies,” *Transactions on Network and Service Management*, April, 2004.
- [2] Barisani, Andrea, “Testing firewalls and IDS with ftester,” In *Insight, Newsletter of the Internet Security Conference*, Vol. 5, 2001, <http://www.tisc2001.com/newsletters/56.html>.
- [3] Bogaerts, Philippe, *HPING tutorial*, August, 2003, http://www.radarhack.com/dir/papers/hping2_v1.5.pdf.
- [4] Eronen, Pasi and Jukka Zitting, “An expert system for analyzing firewall rules,” *Proceedings of the 6th Nordic Workshop on Secure IT Systems*, 2001.
- [5] Eychenne, Herve, *iptables man page*, March, 2002.
- [6] Farmer, Dan and Wietse Venema, *SATAN: Security Administrator's Tool for Analyzing Networks*, 1995, <http://www.fish.com/zen/satan/>.
- [7] Fyodor, “The art of port scanning,” *Phrack*, Vol. 7, Num. 51, September, 1997.
- [8] Gouda, Mohamed G. and Alex X. Liu, “Firewall design: Consistency, completeness, and compactness,” *Proceedings of the International Conference on Distributed Computing Systems*, IEEE Computer Society, March, 2004.
- [9] Internet Security Systems, *Internet Scanner User Guide Version 7.0 SP 2*, 2005, http://documents.iss.net/literature/InternetScanner/IS_UG_7.0_SP2.pdf.
- [10] Lampe, John, *Nessus 3.0 Advanced User Guide*, October, 2005, <http://www.nessus.org>.
- [11] Liu, Alex X., Mohamed G. Gouda, Huibo Heidi Ma, and Anne H. Ngu, “Firewall queries,” *Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS-04)*, LNCS 3544, Springer-Verlag, December, 2004, <http://www.cs.utexas.edu/users/alex/publications/FirewallQueries/query.pdf>.
- [12] Marmorstein, Robert, *ITVal Project Website*, 2005, <http://itval.sourceforge.net>.
- [13] Marmorstein, Robert and Phil Kearns, “An open source solution for testing NAT'd and nested iptables firewalls,” *19th Large Installation Systems Administration Conference (LISA '05)*, pp. 103-112, December, 2005.

QUERY DPORT TO 10.239.202.13 AND FROM 10.239.202.13 AND IN NEW; 1 Port: 25	QUERY DPORT TO 10.239.202.13 AND FROM 192.168.2.3 AND IN NEW; 2 Ports: 22 25	QUERY DPORT TO 10.239.202.13 AND FROM 192.168.2.1 AND IN NEW; 1 Port: 25	QUERY DPORT TO 10.239.202.13 AND FROM 192.168.3.0 AND IN NEW; 1 Port: 25
QUERY DPORT TO 192.168.2.3 AND FROM 10.239.202.13 AND IN NEW; 1 Port: 25	QUERY DPORT TO 192.168.2.3 AND FROM 192.168.2.3 AND IN NEW; 2 Ports: 22 25	QUERY DPORT TO 192.168.2.3 AND FROM 192.168.2.1 AND IN NEW; 0 Ports:	QUERY DPORT TO 192.168.2.3 AND FROM 192.168.3.0 AND IN NEW; 0 Ports:
QUERY DPORT TO 192.168.2.1 AND FROM 10.239.202.13 AND IN NEW; 1 Port: 25	QUERY DPORT TO 192.168.2.1 AND FROM 192.168.2.3 AND IN NEW; 2 Ports: 22 25	QUERY DPORT TO 192.168.2.1 AND FROM 192.168.2.1 AND IN NEW; 0 Ports:	QUERY DPORT TO 192.168.2.1 AND FROM 192.168.3.0 AND IN NEW; 0 Ports:
QUERY DPORT TO 192.168.3.0 AND FROM 10.239.202.13 AND IN NEW; 1 Port: 25	QUERY DPORT TO 192.168.3.0 AND FROM 192.168.2.3 AND IN NEW; 2 Ports: 22 25	QUERY DPORT TO 192.168.3.0 AND FROM 192.168.2.1 AND IN NEW; 0 Ports:	QUERY DPORT TO 192.168.3.0 AND FROM 192.168.3.0 AND IN NEW; 0 Ports:

Figure 21: Queries auto-generated using host classes.

- [14] Marmorstein, Robert and Phil Kearns, “A tool for automated iptables firewall analysis,” *REENIX Track, 2005 USENIX Annual Technical Conference*, pp. 71-82, April, 2005.
- [15] Mayer, Alain, Avishai Wool, and Elisha Ziskind, “Fang: A firewall analysis engine,” *Proceedings of the IEEE Symposium on Security and Privacy*, May, 2000.
- [16] Quinton, Reg, *Using Solaris ipfilters*, <http://ist.waterloo.ca/security/howto/2005-08-19/paper.pdf>.
- [17] Singer, Abe, “Life without firewalls,” *login*, Vol. 28, Num. 6, pp. 27-30, December, 2003.
- [18] Wool, Avishai, “Architecting the Lumeta firewall analyzer,” *Proceedings of the 10th USENIX Security Symposium*, August, 2001.

Secure Mobile Code Execution Service

Lap-chung Lam, Yang Yu, and Tzi-cker Chiueh – Rether Networks, Inc.

ABSTRACT

Mobile code refers to programs that come into a host computer over the network and start to execute with or without a user's knowledge or consent. Because these programs run in the execution context of the user that downloads them, they can issue any system calls that the user is allowed to make, and thus pose a serious security threat when they are malicious. Although many solutions have been proposed to solve the malicious mobile code problem, none of them are truly effective at striking a good balance between defeating zero-day attacks and minimizing disruption to the execution of legitimate applications.

This paper describes a commercial system called SEES that secures the execution of mobile code that comes into a host computer as an email attachment or as a web document downloaded through an anchor link by running them on a separate guinea pig machine rather than on the user machine. Effectively, it takes an isolation approach to the secure mobile code execution problem. As a result, SEES *guarantees* that no malicious email attachments or web documents that act on behalf of the user that downloads them, can damage the resources of the user machine, or can leak any confidential information. In particular, even zero-day virus cannot cause any harms. We present the design, implementation and evaluation of SEES on the Windows platform, and contrast it with other existing approaches to the same problem.

Introduction

Mobile code refers to programs that come into an end user's computer over the network and start to execute with or without the user's knowledge or consent. Examples of mobile code include a Java script embedded within an HTML page, a Visual-Basic script contained in a WORD document, an HTML Help file, an ActiveX Control, a Java applet, a transparent browser plug-in or DLL, a new document viewer installed on demand, an explicitly downloaded executable binary, etc. Because a piece of mobile code runs in the execution context of the user that downloads it, it can issue any system calls that the user is allowed to make, including deleting files, modifying configurations or registry entries, sending emails, or installing back-door programs in the home directory. The most common type of malicious mobile code is email attachment.

Existing solutions to the malicious mobile code problem fall into two categories: signature-based anti-virus tools and behavior blocking tools [1]. Neither of them can stop malicious mobile code or malware effectively. Because existing antivirus tools are based on signatures, there is always a time gap between when a piece of malware first appears and when the corresponding signature is derived and distributed to user sites. For malware such as the SQL Slammer worm, any time gap that is more than a few hours is unacceptable, because a well-designed worm can take down the Internet within hours.

Behavior blocking technology sandboxes the execution of suspicious applications by monitoring and controlling the system calls the applications make, according to a security policy. If properly configured,

behavior blocking can even stop zero-day exploits. However, it is difficult to properly set the sandboxing security policy for each individual application such that all existing applications run smoothly and none of the existing malware can get through. This is especially true when the source code of the applications to be sandboxed is not available. Indeed, almost all existing behavior blocking systems use a single sandboxing policy for all applications running under a user account or on the same system. As a result, these systems tend to trigger many false positives and break perfectly legitimate applications.

This paper describes the design, implementation, and evaluation of a secure mobile code execution system called SEES (Secure Email Execution Service), which takes an *isolation* approach to safeguard an end user machine from two specific types of mobile code, email attachments and documents retrieved via a web browser. SEES identifies incoming email attachments and web documents, and isolates their execution on a physically separate machine (called the guinea pig machine) in such a way that the execution results are displayed on the end user's computer screen with the same look and feel.

Because SEES guarantees that malware embedded in email attachments or web documents never run on an end user machine, it is impossible for them to access, let alone damage, the user's resources. The guinea pig machine itself is carefully configured so that the risks of being permanently compromised and of attacking others when compromised are minimized.

Compared with signature-based anti-virus systems, SEES does not require periodic signature update

and can effectively protect end users from zero-day virus. Even if a piece of malware successfully penetrates and damages the guinea pig machine, the end user's resources still remain intact. Compared with behavior blocking systems, SEES supports fine-grained isolation for specific types of mobile code and uses a physical segregation approach rather than a sandboxing approach. As a result, SEES is less intrusive in that legitimate applications rarely get disrupted because of its security protection.

SEES effectively solves the email virus problem because it addresses the psychological dimension of the problem: People tend to open legitimate-looking but possibly virus-containing email attachments for fear of missing important messages. In addition, more than 90% of virus entering an enterprise is through email. SEES provides an additional level of assurance that even if an email attachment contains virus, it will not be able to inflict any damage upon a user's machine. This ability to tolerate malware allows SEES to reduce the degree of disruption to legitimate application execution to the minimum when compared with other solutions. It also opens up the possibility that a user can experiment with a piece of mobile code, for example, a downloaded executable binary, on the SEES server before she installs it on her own machine.

Related Work

The physical isolation idea of SEES originated from the Spout system [2], which is a distributed execution architecture to secure the execution of Java applets. Spout uses a web proxy to identify Java applets in incoming HTML pages and redirect them to a playground machine. Spout incorporates a Java-based remote display mechanism rather than Windows terminal service for remote execution. A similar approach for secure execution of Java Applets can also be found in [3].

The most widely used tools to protect user machines from malware are antivirus products from Norton [4], McAfee [5], Trend Micro [6]. These antivirus tools can scan files downloaded by web browsers, ftp clients, and email clients in real time. They can effectively remove all known viruses before the viruses can launch the attacks. However, none of them can stop zero-day attacks since they rely on signatures. This deficiency raises a serious security concern because Internet enables malware to spread so fast that even the most prepared antivirus company cannot derive signatures quickly enough to effectively stop them.

Many desktop machines suffer from malware intrusion. Kathleen [7] did a survey on 17 network-based intrusion detection systems, and only Session-Wall from Computer Associates contains a scanner engine to detect viruses embedded in network traffic. However, this scanner still relies on signatures, and therefore cannot handle zero-day exploits. Tripwire [8] is an IDS running on UNIX-like systems that can

detect zero-day viruses. Tripwire computes a hash value for each important system file or binary, and uses it to detect changes to system files. However, it cannot detect viruses that do not modify any system files.

WindowBox [9] from Microsoft is the closest system to SEES. WindowBox implements the sandbox mechanism using a desktop object. WindowBox modifies the Windows 2000 kernel to restrict the access of suspicious applications only to objects created in the same desktop. A user may choose to create many desktops such as work desktop, game desktop and personal desktop, and decide what applications can run on each desktop. If a virus is run on one desktop, it cannot access the network or data created on other desktops.

There are, however, two problems with WindowBox. First, applications running on different desktops still share application configuration files and registries, which a virus may corrupt to cause damage. Second, WindowBox requires users to explicitly decide what applications to run on which desktop and therefore may pose usability problems in practice. In contrast, SEES has neither of these problems because it uses physical isolation and automatic redirection.

Many existing academic sandboxing research systems such as Janus [10], Consh [11], Tron [12], and MAPbox [13] are implemented on top of UNIX-like system. Janus allows users to set permissions on path, environment variables, network access and display. When a process runs under Janus, Janus uses a debugging mechanism to monitor each system call made by the process, and checks the system calls against the user defined security policy. Consh extends Janus by adding a virtual file system and a virtual network system. Besides setting up a security policy, Consh also needs to setup the virtual file system correctly for running applications safely. Instead of setting security policy for a user or a program, Tron allows users to specify different security policies for different instances of the same program. MAPbox groups applications into behavior classes such as editor class, compiler class, mailer class, and browser class, and a special sandbox is built for each behavior class.

Another sandbox example is the secure web browser [14], which is built on top of an OS called SubOS [15], which offers process-specific protection mechanisms. An object downloaded by the web browser is assigned with a sub-userid, and the object is opened or executed on the context of the chosen sub-userid. All of these sandboxing systems require users to set up security policy and choose what and when to sandbox. Such a sandboxing model does not work on Windows environments since many Windows users are used to point and click and they do not have enough computer knowledge to decide what and when to sandbox and setup the sandbox environment correctly. In contrast, the SEES system does not change

the way users use the Windows system. It automatically selects and sandboxes email attachments and web documents.

Secure Mobile Code Execution

A secure mobile code execution service needs to address two fundamental issues: identifying a piece of mobile code when it comes in, and insulating its execution from the host computer that downloads it.

The majority of malware comes into a user's machine because the user clicks on something. Email attachment is the most common channel. Other possible channels include web browsers, ftp programs, peer-to-peer file sharing applications, and messaging applications such as IM and IRC. Another common way through which malware penetrates into a user's computer is by exploiting the automatic download capability of Microsoft's Internet Explorer (IE). Many web pages contain mobile code such as Java scripts, VB scripts, and ActiveX control. If the security level of IE is set to low, IE automatically executes the embedded mobile code when a user visits those pages.

IE also contains vulnerabilities that enable a web page to automatically install a piece of malware on a system even when the security level is set to the highest level. Similar vulnerabilities existed in Microsoft Outlook and Outlook Express. For example, Outlook could execute mobile code contained in an email attachment without a user clicking on it. Finally, by hijacking the control of a server program through such vulnerability as buffer overflow, malware can take over the machine on which the server program runs and potentially spreads to other machines. One such example is the SQL Slammer worm, which exploits a vulnerability in the Resolution Service of Microsoft SQL Server and Microsoft Desktop Engine (MSDE).

The ideal solution to the mobile code identification problem is for each network application to inform the operating system when it downloads and executes a piece of mobile code. Unfortunately neither existing applications nor existing operating systems provide such support. One possible approach to approximate this ideal is to apply binary or source-level program transformation techniques to automatically embed such notification mechanisms into existing network applications without any programming efforts. Because there are a large number of entry points malware can use to infiltrate a Windows PC, SEES chooses to focus on the two most common types of exploit points: email attachments and web objects downloads through IE.

SEES employs API interception techniques to monitor Win32 API calls made by email clients and web browsers, and take proper actions when these programs open or save files. The mobile code identification mechanism used in SEES is independent of email client programs and web browsers.

Once a piece of mobile code is identified, the issue is how to sandbox its execution in such a way

that malware cannot cause damage and legitimate applications can run without any glitches. The key problem here is how to set up the sandboxing policy accurately so as to eliminate both false positives and negatives, and automatically so that the security administration overhead is reduced to the minimum.

Commercial behavior blocking products tend to err on the false positive side in that they tend to apply the same sandboxing policy to all applications executed by a user or on a given machine. An ideal solution to this problem is to apply program analysis techniques to extract application-specific sandboxing policy automatically from arbitrary application programs, for example, the PAID system [16]. However, this approach is not always feasible because the source code of network applications is not always available and introducing such a sophisticated sandboxing mechanism may be impossible for certain user sites.

Traditionally, the scope of sandboxing is a machine or a user account. In the Windows environment, all known behavior blocking systems apply their sandboxing mechanism to all processes running under a user account by limiting their system call privilege. This approach invariably breaks certain benign applications because the sandboxing policy cannot possibly cover the needs of all current and future legitimate applications. For example, the Word program needs read/write access to its application-specific directory and the document directory under the user's home directory. Prohibiting Word from accessing those directories may break the functionality of Word and inconvenience the user.

On the other hand, it is extremely difficult if not impossible to devise a sandboxing policy that can accurately capture and anticipate the requirements of all non-malicious applications such as Word that are going to run on a machine or under a user account. In addition, sometimes even a single process may need to be sandboxed differently at different times. For example, many Windows applications, such as Word, open multiple documents in the same process to reduce resource consumption. This means that when a user opens a local Word document and an email attachment that contains a Word document, she needs to choose between sandboxing both documents and sandboxing neither document.

Instead of sandboxing, SEES chooses to execute mobile code in a different execution environment than the one that downloads it. Specifically, mobile code runs on a physically separate machine called the guinea pig machine under a low-privilege user account, and the result of execution is sent back to the user machine through a remote display mechanism. This execution architecture provides the same look and feel for benign programs but provides physical isolation for potentially malicious programs.

This approach has several advantages. First, it allows centralized management and enforcement of

security policies, and thus reduces administration workload. Specifically, properly configuring the guinea pig machine is all that is needed to defend an enterprise against malicious code embedded within email attachments or web documents. Second, the security policies on the guinea pig machine can be loosened to avoid unnecessary disruption to legitimate applications without compromising security. This additional latitude results from the fact that the guinea pig machine is potentially dispensable and is logically separate from the rest of the intranet.

SEES Implementation

SEES System Architecture

Figure 1 illustrates the system architecture of SEES, which consists of a SEES server and a SEES client. The SEES server runs on a stand-alone machine and provides the isolated execution environment for mobile code. The SEES client takes control when a user opens an email attachment or a web document. Whenever a SEES client needs to open a file that potentially contains mobile code, it sends the file to the SEES server, which opens the file and displays the results on the SEES client's screen. To the user, the look and feel is the same as if the file is opened locally.

The SEES server consists of three components as shown in Figure 2, Execution Manager, Security Control Manager, and System Call Monitor. The Execution Manager allows a SEES client to run a piece of mobile code on the SEES server, and provides the same look and feel as if it is executed locally. The Security Control Manager provides an isolated execution environment so that the side effects of mobile code are completely segregated from the rest of the SEES server. The System Call Monitor is a traditional sandboxing mechanism that protects the SEES server itself from malicious

code by monitoring and controlling system call invocations according to a predefined security policy.

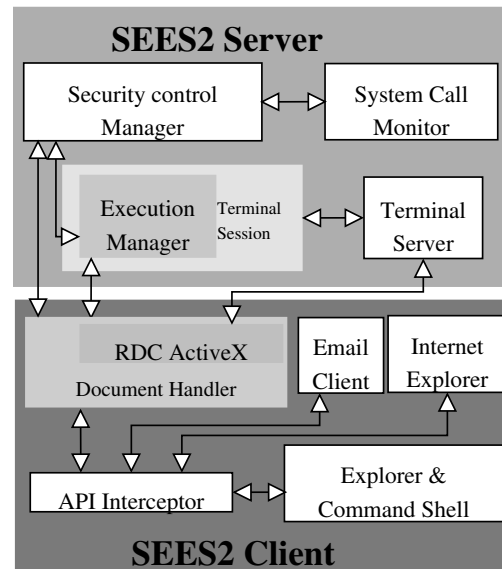


Figure 2: The detailed software architecture of SEES.

The Execution Manager runs a piece of mobile code on the SEES server on behalf of a SEES client. The Security Control Manager provides an isolated execution environment to segregate the side effects of mobile code from the rest of the SEES server. The System Call Monitor protects the SEES server from malicious system call invocations according to a predefined security policy. The main component of SEES client is the API interceptor, which intercepts the save and open operations of application programs and redirects mobile code to the SEES server for execution.

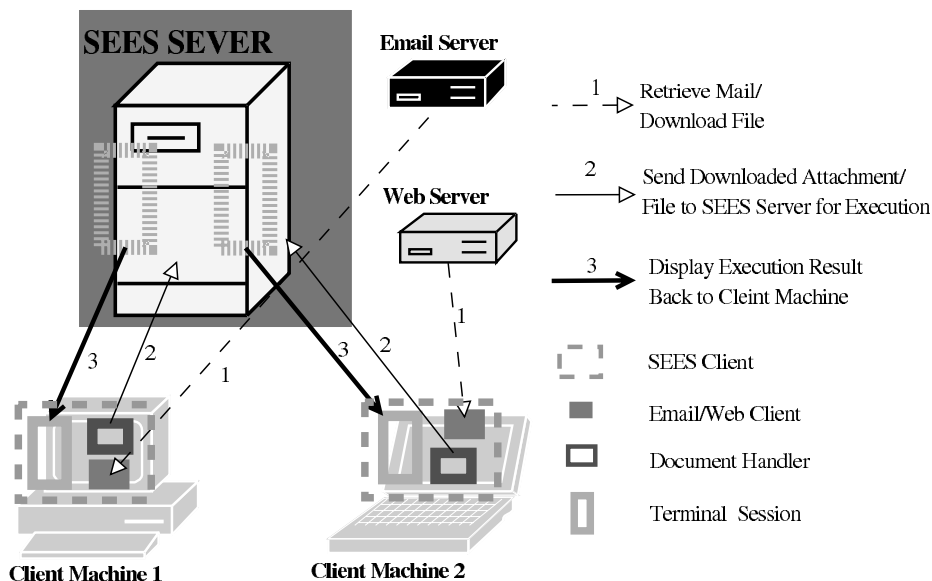


Figure 1: In the SEES architecture, mobile code downloaded from an email client or Internet Explorer runs on a separate guinea pig machine called the SEES server, but the results of mobile code execution are sent back to the end user machine through a remote display mechanism such as Windows terminal service.

The SEES client is implemented on top of Remote Desktop ActiveX Control. When a user invokes a piece of mobile code, the SEES client first consults with the Security Control Manager to obtain a low-privilege account, and executes the mobile code under this account on the SEES server. There are several implementation alternatives to supporting the remote execution mechanism, including Windows terminal server, Linux server running Wine or Crossover Office and VNC, and Windows server running multi-user VNC or X Windows.

The Windows terminal server is the best choice in term of performance overhead and usability, but it requires expensive licensing charge. A less expensive way is to use a Linux server running Wine and VNC (LWV), which is slower and requires much more memory. Currently Wine can successfully run many Windows applications such as Microsoft Office. Still, there are many other Windows applications that cannot run under Wine. Finally, we have developed an experimental version of multi-user VNC for the Windows platform, but its performance for interactive applications is still inferior to Windows terminal server.

Mobile Code Identification

The main task of the SEES client is to identify potentially dangerous contents downloaded from the network and send the contents to the SEES server when users invoke them. Since mobile code can have many different forms, and they can come into a computer from many channels, there is no universal mechanism that can identify mobile code accurately. Our current approach is to treat the files downloaded by Internet Explorer or an email client and with dangerous MIME type such as .exe and .doc as dangerous contents.

The easiest way to identify downloaded contents is to use a proxy server to monitor and parse the incoming contents, and mark the contents as dangerous if the contents have certain MIME type. The first version of SEES used this approach. More concretely, a POP3 proxy server is used to intercept all incoming emails and rename an email attachment if it contains dangerous MIME type such as .exe and .doc. The POP3 proxy server adds a .sees extension to each dangerous attachment so that when a user clicks on a renamed attachment, the SEES client is invoked instead of the corresponding application.

However, this proxy server approach has two major drawbacks. First, the .sees extension tends to create confusion because it is visible to the end users. Second, many email servers, such as Microsoft Exchange Server and IBM Lotus Domino, use different protocols between themselves and email clients. Worse yet, emails could be encrypted or email servers could require a secure connection. As a result, the proxy server approach inherently entails significant implementation complexity that cannot be easily removed.

To avoid these problems, SEES employs a client-side Win32 API interception mechanism to identify

email attachments and downloaded web documents. The software architecture of the SEES client is shown in Figure 2. Specifically, the SEES client intercepts file open and file save operations of email clients and IE. When the user double-clicks on an email attachment or a link on an HTML page, the email client or IE calls the ShellExecute family of API to open the attachment/file. The API interceptor intercepts the ShellExecute family of API calls and re-directs the attachment/file to the SEES server. If the user attempts to save an attachment/file to a target file, the API interceptor intercepts the GetSaveFileName family of API calls to append an .sees extension to the target file if it is in a FAT file system, or flags a “dangerous” flag of the target file if it is in an NTFS file system.

The .sees extension and the unused flag are meant to indicate to the system that the file is potentially dangerous. Later on, when a user opens this file through the Windows Explorer or command shell, the IShellExecuteHook component intercepts the ShellExecute family of API calls, examines whether the file is dangerous, and re-directs it to the SEES server if it is dangerous. The advantage of using API interception to identify downloaded files is that a single mechanism can work with many different applications as long as they use the ShellExecute family of API. Currently, we are extending this interception mechanism to the peer-peer applications and instant messenger applications.

API Interception

When an email client or IE opens a file, the Win32 API interceptor intercepts the file open call and sends a request to the SEES server to open it. If an email client or IE needs to save a file, the interceptor marks the file as dangerous. When a user opens a dangerous file, the interceptor also sends the file to the SEES server. The ShellExecute family of APIs in shell32.dll of the Windows platform are the most commonly used APIs to perform file operations on a file. Although they are not the only APIs that can be used to open a file or to execute a file, they are indeed used by all email clients we have tested, as well as IE, Explorer, and the DOS shell.

Intercepting Win32 API calls means taking over the program control when these APIs are called without modifying the monitored applications. The most commonly used interception mechanisms are *Proxy DLL*, *EAT Patching*, *IAT Patching*, and *Shell Extension*. Proxy DLL replaces an original DLL with a proxy DLL that contains a call stub for each exported function in the original DLL. The replacing proxy DLL assumes the name of the original DLL, while the original DLL is renamed.

When an application uses the original DLL's name to load a DLL, it is the proxy DLL that gets loaded instead. All the calls made to the functions in the original DLL are routed to the exported stubs in the proxy DLL. The proxy DLL can simply forward the calls to the original DLL, perform some operations

before forwarding, forward the calls to someone else, or reject the calls. Proxy DLL is the simplest way to intercept Win32 APIs. However, this technique requires that the function prototypes of all the exported functions in a DLL be available. SEES does not use this approach because some function prototypes of the DLLs we want to intercept are not available.

The Portable Executable (PE) format [17] is the binary format used by both executable files and DLLs on the Windows platform. Each DLL PE file contains a table called *Export Address Table* (EAT) that stores the entry point of each exported functions. The addresses stored in a DLL's EAT are used for an application to call the functions the DLL exports. To intercept a function exported by a DLL, one can add to the DLL file a new section to store the intercepting function's code, and modify the EAT entry of the intercepted function to point to the intercepting function.

When an application calls an intercepted function, the intercepting function is activated, and the intercepting function can choose to abort the function call, forward the function call to the original function, or perform some other operations. However this EAT patching technique cannot be easily applied to system DLLs since the Windows File Protection (WFP) mechanism discourages such DLL modifications by nullifying the effects of these modifications. Even if all backup versions of a system DLL are replaced, Windows OS can still restore the DLL to the unmodified version through the Windows Updates mechanism. Another problem of this approach is that modification to a DLL has to be compatible with future versions of the DLL.

Each executable PE file includes an *Import Address Table* (IAT), which has an entry for each imported function (a function exported by a DLL). After an executable file is loaded into memory, this table is filled with the addresses of the imported functions. When an application makes a call to an imported function, it first looks up the corresponding IAT entry, and then uses the address contained within to jump to the target function. IAT Patching modifies the IAT entry of an intercepted function to point to the new intercepting function. All intercepting functions are implemented in a DLL, and the intercepting functions are loaded into memory by a method known as *DLL injection*.

There are three ways to inject a DLL to the address space of a running process: 1) using the Win32 API `SetWindowsHookEx`, 2) using the Win32 API `CreateRemoteThread`, and 3) using the `Applnit_DLLs` registry. All these three mechanisms force the Windows OS to load a specified DLL automatically. Each Windows DLL has a function called `DllMain`, which is called automatically by the Windows OS after a DLL is loaded. Therefore, the code for patching the IAT table can be implemented in `DllMain`. However, this IAT patching technique only works with statically loaded

DLLs. A DLL can be loaded dynamically by using `LoadLibrary` and the entry point of a function exported by a DLL can be obtained by `GetProcAddress`. To support dynamically loaded DLLs, both `LoadLibrary` and `GetProcAddress` must also be intercepted via IAT patching.

The `IShellExecuteHook` interface is a shell extension that can intercept any calls made to `ShellExecute` (EX). This is the documented approach to extend the behavior of the `ShellExecute`(EX) API with low overhead, but it cannot intercept other Win32 API calls. SEES uses both IAT patching and the shell extension method to intercept the `ShellExecute` family of APIs and other APIs. The `ShellExecute` family of APIs sometimes are used to perform some other operations beside opening an attachment. The SEES client analyzes the arguments used in these API calls to filter out unwanted cases.

Saving Files to Local Disk

Even though mobile code runs on the SEES server, it is essential that the user feels that it is executed locally. Towards that end, when a user attempts to save a file from an application running on the SEES server, the file save interface should show the file system on the user's machine rather than that on the SEES server. To implement local save for a remote execution mechanism such as Windows Terminal Server, one needs to re-direct the file save operation from the SEES server to the requesting SEES client. More concretely, when the Execution Manager intercepts a save operation from applications running on the SEES server, it requests the SEES client to launch a save as dialog on the SEES client machine, thus providing the illusion that the user is saving the attachment/file on the local disk.

After the user picks a local file name for the save operation, the Execution Manager first stores the file on the server's disk, and then transfers it to the SEES client, which then saves the copy to the file location the user specifies. However, since the SEES server cannot always detect if an opened attachment is a virus or not, after the user saves the attachment to the local machine, it is also marked as dangerous so that it will be sent to the SEES server next time when the user clicks on it again.

Isolation of Mobile Code Execution

Because the SEES server is responsible for executing mobile code on behalf of all SEES clients within an organization, it is essential to protect it from malicious mobile code. That is, it should not be possible for any mobile code to bring down the SEES server and deny the mobile code execution service to other hosts. To provide such protection, SEES adds the following checks for every mobile code execution request from the SEES clients:

- Only certain IP addresses are authorized to be a SEES client.
- Each SEES client can only make a finite number of mobile code execution requests.

- The total amount of memory and disk usage by a SEES client is limited.

The SEES server executes each piece of mobile code on a low-privilege account, which allows its processes to read/write its home directory and to have read access to certain system applications and files. As a result, no mobile code can steal information from the SEES server or corrupt the system data structures such as registries, DLLs, and applications. To prevent mobile code from leaving any permanent effects on the SEES server, all the modifications to the registries and file system made by a piece of mobile code are erased after the execution is done.

More specifically, after installation, the SEES configuration tool copies the registry files NTUSER.DAT and UsrClass.dat of each account into a safe place. After execution of each piece of mobile code, these two registry files are restored automatically. As a result, when a piece of mobile code starts, it always start with a “clean” execution context, in terms of registry values and home directory contents, and will never get “infected” by other malicious mobile code.

A key advantage of the SEES architecture is that the SEES server can use a more lenient security policy when executing mobile code, as long as such policies never bring down the server. That is, the SEES server only needs to protect itself from denial-of-service (DOS) attacks, and can afford to err for other types of attacks, since the server itself is not supposed to contain any valuable information. Because of this additional latitude, SEES is much better than existing behavior blocking systems because it can minimize disruption to legitimate applications.

To take this idea to the extreme, we are currently exploring a *namespace virtualization* mechanism that allows each piece of mobile code to modify whatever files and registries it wants, and yet these modifications are never visible to the SEES server or other pieces of mobile code. This mechanism ensures no legitimate mobile code will be disrupted while protecting the SEES server from malicious mobile code.

System Call Monitoring

As an additional layer of defense, the SEES server also includes a system call monitor that checks all the system calls made during the mobile code execution against a pre-defined sandboxing policy. Any system calls that violate the sandboxing policy are denied, and an alert message is sent to the user.

The software architecture of SEES’s system call monitor is shown in Figure 3. The current SEES prototype monitors only two system calls, NtOpenFile and NtCreateFile, which are used for both files and network connection operations. To intercept system calls on a Windows NT-like environment, we modify KeServiceDescriptorTable [18], which is the system call dispatch table data structure in the kernel. The System Call Interception module first changes the table entries corresponding to NtOpenFile and NtCreateFile to point to two SEES hooking functions respectively and saves the original function pointers. Consequently, all calls to NtOpenFile and NtCreateFile go through SEES’s hooking functions.

When the hooking functions intercept a system call, the Call Source Identification module needs to identify the process that makes the system call, since only processes that execute mobile code and the child processes they create need to be sandboxed. This

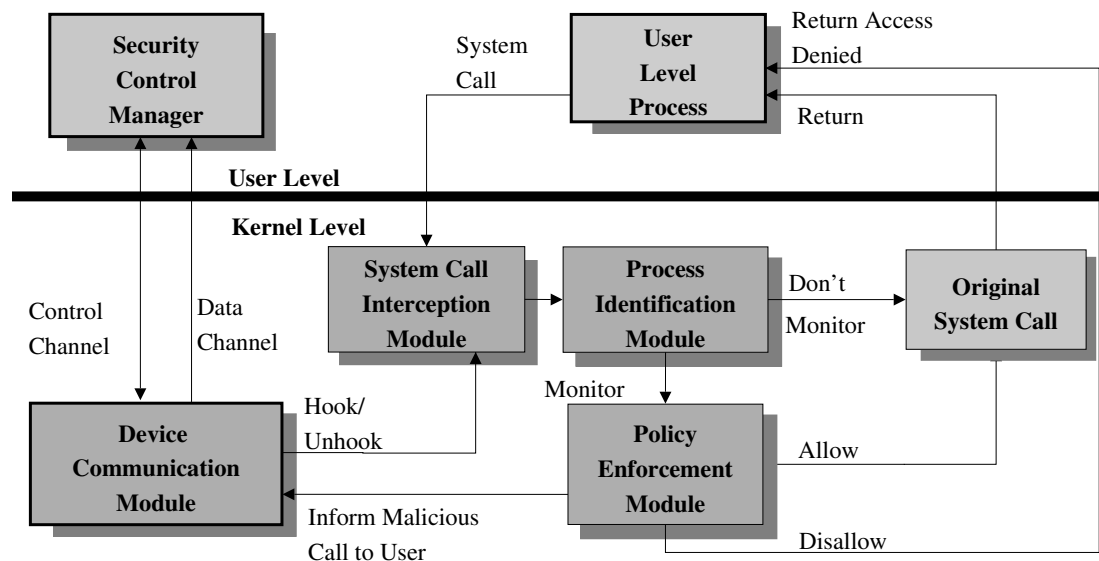


Figure 3: The detailed software architecture and control flow of the System Call Monitor driver, which is embedded into the kernel. The System Call Monitor only sandboxes the applications that run under a low privilege SEES user account according to the security policy set by a system administrator. It allows the system applications and the applications that are not executed under a SEES user account to run normally.

module obtains all the relevant information, such as process ID, terminal session ID, full path of the binary image of the current process, security identifier (SID) of the current user, filename or object name (network object) to be opened, and desired access mode, from the process structure of the current process making the system call. The Policy Enforcement module then uses the collected information to check for access violation. These policy rules are set up at the configuration time and loaded into the system call monitoring driver during initialization. The Device Communication module is responsible for informing the user when a sandboxed process violates the security policy.

Evaluation

Scalability

The main scalability concern about SEES architecture is the fact that all mobile code's execution is concentrated on a single server using Windows Terminal Services (WTS). In this subsection, we will examine the start-up latency of individual applications and CPU/memory consumption on the SEES server. To be sure, WTS actually supports server clustering to improve overall throughput and fault tolerance. We use a DELL desktop with Intel Pentium 4 2.4 GHz CPU, 768 MB PC2700 DDR memory as the SEES server and an Acer desktop machine with Intel Pentium 3 650 MHz CPU and 384 MB PC133 SDRAM memory as a SEES client. The operating systems used on both machines are MS Windows 2000.

We tested different types of documents using applications in Microsoft Office Suite and the result shows that when the available physical memory on the SEES server is more than 20 MB, the start-up latency between local execution and SEES-model execution is only about one second on average. When the number of active terminal sessions increases, e.g., increasing to 30 or 40 sessions, there is still no noticeable latency difference between the two cases. When the available physical memory is below 20 MB, the application startup latency increases significantly, i.e., 5 seconds or longer, because of extensive swapping. This means the architecture based on WTS does not add to additional usability problem in terms of latency as long as the SEES server is installed with enough memory.

The SEES server's CPU usage is also related to its memory consumption. When the SEES server's available physical memory is more than 20 MB, its CPU usage usually reaches a peak value between 50% and 90% when a terminal session starts or terminates, but quickly decreases to a lower average value. However, after the available physical memory becomes smaller than 20 MB, the CPU usage remains at a high peak value and that is when the start-up latency starts to deteriorate.

When the number of active terminal sessions increases, the memory usage on the SEES server increases linearly, as shown in Figure 4. In this test, we created a series of new terminal sessions, each running a WORD instance that opens a 865KB document,

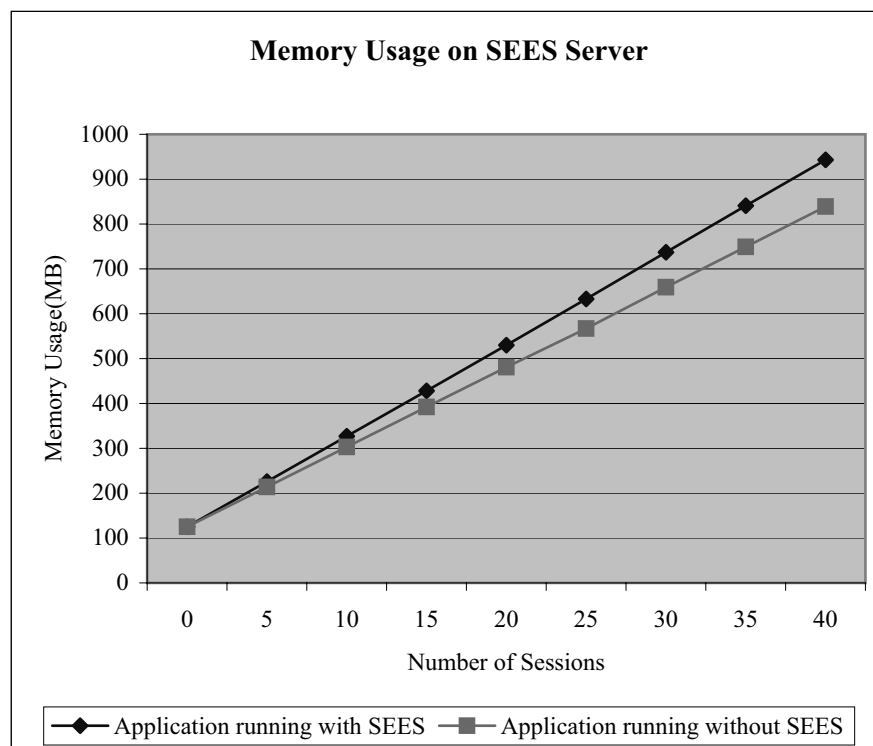


Figure 4: The physical memory consumption of the SEES server increases linearly at a rate of 20 MB per new terminal session. In this case, each terminal session runs a WORD instance that opens a 865KB document.

and the rate of increase in the SEES server's memory consumption is about 20 MB per new session. To isolate the contribution of the SEES server, we removed the SEES server control and opened the same WORD document in new terminal sessions, and the increment in physical memory usage is about 18 MB per session. This shows that the SEES server itself consumes only about 2 MB. The WORD application in this case consumes 8 MB because it opens a document containing many images. This means that there is a fixed overhead of about 10 MB associated with each new terminal session even when it is not running any application, and a remote-display scheme better than WTS can be exploited to reduce the memory overhead.

Attack Analysis

Saving Attachment Directly From Email Client

Instead of opening an email attachment, users can also save an email attachment directly from an email client application. This operation is different from the "Local Save" functionality of SEES and must be identified properly. One solution is to intercept this operation of email client applications and mark the file to be saved.

Time-bomb Malicious Code

A time-bomb malicious code will not be triggered until some later time or when a specific user operation takes place. As a result, the system call monitor on the SEES server may not immediately detect a time bomb's malicious behavior. One way to resolve this problem is to mark the file saved through "Local Save" so that all subsequent invocations of these files will still be executed on the SEES server.

Benign Installer and Malicious Executable

It is difficult to distinguish a benign application installer from a malicious executable, because both can modify system directories and registries. As a result, the SEES server may falsely mistake a legitimate application installer for malicious mobile code. We are working on a namespace virtualization technique that provides a process its own virtual execution environment by logging all its updates to system resources, and committing these updates only when it is sure of the process's legitimacy.

User Context Contamination

Because a malicious email attachment can always update the private registry entries or home directory of the user account under which it runs, these updates can potentially infect future email attachments that execute under the same user account. To address this problem, the SEES server provides an account refreshing mechanism, which cleans up the home directory and refreshes the private registry entries as soon as an existing session terminates.

Attacks Against the SEES Server

The home directory of a SEES user account on the SEES server is configured to be writable for processes under that user account. If a malicious email

attachment keeps creating files, it may consume excessive disk space on the SEES Server. To prevent this attack, SEES sets a disk quota limit for each account using NTFS's quota management. If an attacker somehow gets hold of the user ID and password of a user account on the SEES server, she can bombard the SEES server with many terminal sessions. The SEES server solves this problem by limiting the total number of terminal sessions per host. In addition, the attacker can log into the terminal server to eavesdrop the current applications running under the same user account or to browse related network shares. To stop such attacks, the SEES server ensures that terminal sessions under the same user account always come from the same host, and the network share component is disabled.

Conclusion and Future Work

SEES stands out among both research and commercial solutions to the secure mobile code execution problem because of its unique capability of both stopping zero-day virus and minimizing disruption to execution of legitimate applications. It achieves this through accurate identification of specific types of mobile code and physical isolation of the execution of these mobile code. The end result is that SEES can guarantee that no email attachments and web documents can act on behalf of the user that downloads them, can damage the resources of the user machine, or can leak any confidential information.

As we discussed in the section on the fundamental issues of secure mobile code execution, there are many other mobile code entry points that the current SEES implementations do not capture and therefore cannot isolate. For example, it is difficult to identify and sandbox mobile code embedded in an HTML page or an email body, especially when the HTML page or email is encrypted.

The main problem is that this type of mobile code runs in the same address space as the downloading application, in this case Internet Explorer or Outlook, and requires the sandboxing mechanism to use different sandboxing policies at different times for the same application. As another example, mobile code embedded within objects being exchanged through FTP applications, peer-to-peer file sharing applications, IRC and Instant Messaging applications becomes increasingly prevalent, and thus needs to be identified and sandboxed properly.

Although the physical isolation approach in the current SEES implementations provides strong protection, it has two disadvantages. First, it requires an expensive infrastructure, namely the Windows Terminal Server. Second, it cannot be easily generalized to a mobile computing environment, because the SEES server infrastructure may not always be available. To address these problems, we are currently developing a logical isolation approach that relies on system call

monitoring and virtualization techniques, and thus does not require a separate guinea pig machine. We expect this approach to be more scalable and portable while providing the same degree of protection as the physical isolation approach.

Author Biographies

Dr. Lap Chung Lam is the Chief Engineer of Rether Networks Inc. He received his B.A. in CS and mathematics from SUNY at New Paltz, and Ph.D. in CS from Stony Brook University in 1997 and 2005 respectively. He received a best paper award from the Program Analysis for Security and Safety Workshop (PASSWORD) co-located with ECOOP 2006. Dr. Lam's current research interest focuses on computer security, software protection, and program analysis. He can be reached electronically at lciam@rether.com.

Yang Yu is a Ph.D. candidate in the Computer Science Department of Stony Brook University. He received his B.S. and M.S. in computer science from Tsinghua University, Beijing, China in 1999 and 2002 respectively, and M.S. in computer science from Stony Brook University in 2005. He has received a Best Paper Award from 2005 Annual Computer Security Applications Conference (ACSAC). His current research interest lies in operating system and system security. He may be reached at yyu@cs.sunysb.edu.

Dr. Tzi-cker Chiueh is a Professor in the Computer Science Department of Stony Brook University, and the Chief Scientist of Rether Networks Inc. He received his B.S. in EE from National Taiwan University, M.S. in CS from Stanford University, and Ph.D. in CS from University of California at Berkeley in 1984, 1988, and 1992, respectively. He received an NSF CAREER award in 1995, an IEEE Hot Interconnect Best Paper award in 1999, a Long Island Software Award in 1997 and 2004, and a Best Paper Award from 2005 Annual Computer Security Applications Conference (ACSAC). Dr. Chiueh has published over 140 technical papers in refereed conferences and journals. His current research interest lies in wireless networking, computer security, and storage systems.

Bibliography

- [1] Conry-Murray, Andrew, *Product focus: Behavior-blocking stops unknown malicious code*, 2002, <http://www.networkmagazine.com/shared/article/showArticle.jhtml?articleId=8703363&classroom=>.
- [2] Chiueh, Tzi-cker, Harish Sankaran and Anindya Neogi, "Spout: A transparent distributed execution engine for java applets," *IEEE Journal of Selected Areas in Communications*, Vol. 20, 2002.
- [3] Malkhi, D. and M. K. Reiter, "Secure execution of java applets using a remote playground," *IEEE Transactions on Software Engineering*, Vol. 26, 2000.
- [4] Symantec: *Norton antivirus 2004 professional*, 2004, http://www.symantec.com/nav/nav_pro/features.html.
- [5] McAfee: *McAfee virusscan*, 2004, <http://us.mcafee.com/root/package.asp?pkgid=100>.
- [6] Trend Micro: *Officescan*, 2004, <http://www.trendmicro.com/en/products/desktop/osce/evaluate/features.htm>.
- [7] Jackson, Kathleen A., "Intrusion detection system (ids) product survey," *Los Alamos National Laboratory report LA-UR-99-3883*, 1999.
- [8] Kim, G. H. and E. H. Spafford, "The design and implementation of tripwire: A file system integrity checker," *ACM Conference on Computer and Communications Security* pp. 18-29, 1994.
- [9] Balfanz, Dirk and Danie R. Simon: "Window-box: a simple security model for the connected desktop," *Proceedings of the 4th USENIX Windows Systems Symposium*, pp. 37-48, 2000.
- [10] Goldberg, Ian, David Wagner, Randi Thomas, and Eric A. Brewer, "A secure environment for untrusted helper applications," *Proceedings of the 6th USENIX Security Symposium*, San Jose, CA, 1996.
- [11] Alexandrov, Albert, Paul Kmiec, and Klaus Schauer, "Consh: A confined execution environment for internet computations," *USENIX Annual Technical Conference*, 1999.
- [12] Berman, Andrew, Virgil Bourassa, and Erik Selberg: "Tron: Process-specific file protection for the UNIX operating system," *Proceedings of the 1995 USENIX Technical Conference*, pp. 165-175, 1995.
- [13] Acharya, Anurag and Raje Mandar, "Mapbox: Using parameterized behavior classes to confine untrusted applications," *Proceedings of the Tenth USENIX Security Symposium*, 2000.
- [14] Ioannidis, Sotiris and Steven M. Bellovin, "Building a secure web browser," *USENIX Annual Technical Conference, FREENIX Track*, pp. 127-134, 2001.
- [15] Ioannidis, Sotiris and Steven M. Bellovin, "Sub-operating systems: A new approach to application security," *Technical Report MS-CIS-01-06*, University of Pennsylvania, 2000.
- [16] Chiueh, Tzi-cker, *Paid: Program-semantics aware intrusion detection*, 2003, <http://www.ecsl.cs.sunysb.edu/paid/index.html>.
- [17] Microsoft Corporation, *Microsoft portable executable and common object file format specification*, 1999, <http://www.microsoft.com/whdc/hwdev/hardware/PECOFF.msp>.
- [18] Schreiber, Sven B., *Undocumented Windows 2000 Secrets A Programmer's Cookbook*, Addison-Wesley, pp. 266-268, 2001.

FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs

Adam Slagell, Kiran Lakkaraju, and Katherine Luo
– NCSA, University of Illinois at Urbana-Champaign

ABSTRACT

FLAIM (Framework for Log Anonymization and Information Management) addresses two important needs not well addressed by current log anonymizers. First, it is extremely modular and not tied to the specific log being anonymized. Second, it supports multi-level anonymization, allowing system administrators to make fine-grained trade-offs between information loss and privacy/security concerns. In this paper, we examine anonymization solutions to date and note the above limitations in each. We further describe how FLAIM addresses these problems, and we describe FLAIM's architecture and features in detail.

Introduction

As computer systems have become more interconnected and attacks have grown broader in scope, forensic investigations of computer security incidents have crossed more and more organizational boundaries [7]. This poses a difficulty for the computer security engineer since it becomes more difficult to understand attacks from the narrowing perspective they have from the vantage point of just their own logs. Consequently, there is an increased desire to share logs within the security operations community.

This increased demand is clearly seen within the community of security operations, but developers, researchers and educators also depend upon log sharing. Developers of forensic and log analysis tools need records from real incidents to test the effectiveness of their new tools. Networking researchers depend heavily upon large and diverse data sets of network traces [13]. Security and honeynet researchers also desire large and diverse data sets of logs and network traces to do their research [19]. Educators in traditional academia as well as those that train security engineers (e.g., the SANS Institute) depend upon real-life examples for students to analyze and incorporate into assignments. Consequently, in step with the growth of the computer security field, there has been an increase in the need for sound methods of sharing log data.

While it is generally recognized that sharing logs is important and useful [16], it is very difficult to accomplish even among small groups. The difficulty arises because logs are sensitive, and it is difficult to establish high levels of trust between multiple organizations – especially in a rapid manner in response to distributed, yet related attacks. The consequences of logs getting into the wrong hands can be severe whether they are simply mishandled by friendly peers or fall directly into the hands of adversaries through public disclosure.

There are many types of potentially sensitive information in logs, and as such, logs are like a

treasure map for would-be attackers. Access to them can provide special views of security weaknesses not visible from outside scans, or in the very least make the attacker's reconnaissance job much simpler. Logs may reveal bottlenecks as potential DoS targets, record plaintext login credentials, reveal security relationships between objects or reveal soft targets (e.g., machines already compromised and part of an existing botnet). For these reasons, security administrators are reluctant to share their logs with other organizations, even if there is a potential benefit.

While a draconian vetting process for would-be recipients and strong physical controls over the location of log data could be used to address some of these concerns about sharing sensitive logs, such techniques are not flexible or scalable. Anonymization is an eminently more flexible solution, and it has been increasingly employed in recent years. Still, the tools and methods to-date that anonymize logs are immature, and log sharing has not been achieved at the levels expected or with the ease desired. It is true that there are log repositories out there, but they all seem to share at least one of the following problems:

1. They do not have a wide view of the Internet, but they are quite localized;
2. the repositories are very specific, addressing only one or a few types of logs;
3. anonymization techniques are weak – often nonexistent – and usually inconsistent between submissions; or
4. they collect many logs but do not share them with the research community [16].

Furthermore, even if someone is willing to take a risk and submit a large number of logs for public consumption, security engineers are still faced with the difficulties of gathering specific logs from other organizations during an on-going investigation.

We contend that the problem stems from the fact that tools for log anonymization are immature and not able to meet the many, varied needs of potential users.

Specifically, they have been one-size-fits-all tools, addressing only one type of log, often anonymizing only one field in one way. We desperately need more flexible tools that are

1. highly extensible;
2. multi-level, supporting many options for each field, allowing one to customize the level of anonymization for their needs;
3. multi-log capable, being flexible enough to support the anonymization of most security relevant logs without major modification; and
4. have a rich supply of anonymization algorithms available for use on various fields.

To meet these needs, we have developed FLAIM, the Framework for Log Anonymization and Information Management, a C++ based anonymization tool for popular UNIX-like operating systems (e.g., Linux, FreeBSD, OpenBSD, NetBSD and Mac OS X).

FLAIM strictly separates parsing from anonymization, providing an API through which run-time dynamically loaded parsing modules can communicate with the rest of the framework. Details such as file I/O are abstracted away from FLAIM's core, making it possible to handle streamed data as easily as static data on the disk. The anonymization engine, which consists of a suite of anonymization primitives for many different data types, is also separated from the profile manager which manages the XML based anonymization policies (e.g., parsing policies, validating policies against schemas, etc.). These three components, together, provide an extensible and modular anonymization framework able to anonymize data to multiple levels for multiple types of logs.

While we have only talked about computer and network logs to this point, the uses for a general framework for anonymization extend far beyond the sharing of network logs. In late 2004, University of California – Berkeley researchers were denied permission to analyze a large set of personal data about participants in a state social program after their systems were hacked and data on approximately 1.4 million people were breached [12].

This example highlights two points. First, research often depends on access to large amounts of data. The UC-Berkeley team was investigating how to provide better care to homebound patients. Similarly, proponents of an NSA domestic spying program have claimed that the 9/11 hijackers could have been identified by a program that analyzed communications data [5]. Second, there are concerns of privacy protection on these data sets seen in the backlash against such programs.

Anonymization potentially offers the best of both worlds, allowing analysis while also protecting privacy. If the UC-Berkeley data had been anonymized before being distributed, the vulnerability to identity theft and other misuse may have been mitigated. While we have not created modules for these other data sets yet in

FLAIM, we have made FLAIM's core fairly agnostic about the data source, capable of working with any sort of record/field formatted data. This generality opens up many possible future applications that we have not even considered as of yet.

The rest of this paper is structured as follows. In the next section, we present an overview of FLAIM, discussing its goals and functionality. We then present a detailed description of the FLAIM architecture and API followed by a description of the anonymization algorithms available in FLAIM. We follow with an extended example of their use and a discussion of other log anonymization tools. We then conclude and discuss future work.

Overview of FLAIM and Goals

There are four properties that an anonymization tool must have in order to meet the varied needs of potential users. The anonymization tool must: supply a large and diverse set of anonymization algorithms, support many logs, support different levels of anonymization for a log, and finally have an extensible, modular architecture.

These properties are not independent but to a large extent depend upon each other. Namely, to support multi-level anonymization of logs it is necessary to have multiple types of anonymization algorithms. In order to provide support for multiple types of logs it is extremely useful to have an extensible, modular architecture for the tool.

Utility and Strength of Anonymization Primitives

We use the term *anonymization algorithm* or *anonymization primitive* to describe an algorithm that takes as input a piece of data and modifies it so that it does not resemble its original form. For instance, we could have an anonymization algorithm that takes proper names, such as "Alice" and "John" and transforms each name by constructing an anagram (e.g., "John" becomes "Honj" and "Alice" becomes "Cliea").

Clearly, anyone seeing these anonymized names would be able to guess at the unanonymized names. Thus, we define the *strength* of an anonymization algorithm to be related to the difficulty of retrieving information about the original values from the anonymized values. For instance, a black marker anonymization algorithm, which just replaces every name with the string "NULL" is very strong – requiring one to use only other identifying fields if they want to gather information about the unanonymized value of that field. In a similar way, we can talk about the strength of a set of anonymization algorithms applied to several distinct fields. For example, a scheme that only anonymizes one field might not be strong enough, but one that anonymizes a set of three fields together may be much stronger.

The strength of an anonymization algorithm is crucial to making sure that private data is not retrieved from shared logs, but this is not the sole goal of sharing

logs. The real purpose to sharing logs is to allow analysis of your logs. To do this, one must preserve some type of information in the logs. The *utility* of an anonymization algorithm is related to how much information is preserved in the anonymized value. Note that just as we can talk about the security of an anonymization primitive, we can also talk about the utility of a scheme or set of algorithms applied to specific fields.

Consider the proper name anonymization example from above. The anagram anonymizer has a greater utility than the black marker anonymization primitive. With the anagram anonymized names, we lose the structure of the word, but there still remains the number of letters and the actual letters from the unanonymized value. The black marker anonymization algorithm strips away all such information.

It is clear that the utility and strength of an anonymization algorithm are strongly related. Furthermore, these relationships are complex, since the utility and strength are based on the type of analysis we are doing. In our future work we plan on further exploring the relationship between the utility and strength of various anonymization algorithms based on the task of detecting security problems in network logs. However, a first step to reaching this goal is creating a flexible tool like FLAIM that allows us to make such trade-offs in anonymization.

Diverse Set of Anonymization Algorithms

As we mentioned earlier, the central goal of sharing network and system logs is to aid in the analysis of security related incidents on the network. And essential to meeting this goal for various organizations and their unique scenarios is the ability to make trade-offs between the *utility* of the anonymized log and the *strength* of the anonymization scheme. A necessary condition to make these trade-offs is to have a diverse set of anonymization primitives available for the tool at hand.

Earlier, we discussed how different anonymization primitives can make different trade-offs between security and information loss with a simple example. Now let us examine a more complex example with IP addresses. Table 1 shows a set of IP addresses anonymized by various algorithms:

- Black marker anonymization, in this example, maps all the IP addresses to the same value, 10.1.1.1. This leaves no scope for analysis on that field.

- Random permutation maps each IP address to another address at random. This allows some type of analysis, as we could determine if two hosts in different records are the same.
- Truncation, in this example, removes the last 16 bits of an IP address. This allows one to see what the different domains are, but several hosts are collapsed down into single values. Thus, it becomes difficult to separate individual hosts in the records.
- Finally, prefix-preserving anonymization maps the IP addresses to random addresses, but it preserves the subnet structures. In this example, we see that the 141 class A network is consistently mapped to 12, preserving the subnet structures, but not revealing the original subnets. In addition, it preserves individual hosts like a random permutation, thus preserving much more utility than the other methods. However, this comes at a cost of anonymization strength as we have shown in [16].

Similar trade-offs can be made for other fields by having multiple anonymization algorithms available. Even the ability to decide which fields are anonymized, provides a mechanism to make similar trade-offs. However, having this vast set of anonymization primitives is just one of the necessary conditions to create multiple levels of anonymization, a point we discuss further in a later part of this section.

Supporting Multiple Logs

A security incident is often only revealed in the presence of multiple logs. Thus, proper analysis often requires access to many different logs. Therefore, a holistic anonymization solution should anonymize different types of logs. While one could write a separate tool for each log type, this is a very inefficient approach. This is especially apparent when we notice that many logs share the same set of common fields (e.g., IP address, port number, timestamp). The only difference is often in how these fields are represented within the logs (e.g., dotted decimal IP addresses in netfilter logs and binary 32 bit unsigned integers in network byte order in NetFlows).

FLAIM supports multiple logs by having many “modules”; each module parses one specific type of log. New modules can be created and added to FLAIM by merely implementing a simple interface that

IP	Black Marker	Random Permutation	Truncation	Prefix Preserving
141.142.96.167	10.1.1.1	141.142.132.37	141.142	12.131.102.67
141.142.96.18	10.1.1.1	141.142.96.167	141.142	12.131.102.197
141.142.132.37	10.1.1.1	12.72.8.5	141.142	12.131.201.29
12.161.3.3	10.1.1.1	212.3.4.1	12.161	187.192.32.51
12.72.8.5	10.1.1.1	141.142.96.18	12.72	187.78.201.97
212.3.4.1	10.1.1.1	12.161.3.3	212.3	31.197.3.82

Table 1: Example of four common methods to anonymize IP addresses.

communicates through the FLAIM API. Modules are loaded as run-time libraries, so no recompilation of FLAIM is necessary. This allows users to leverage the diverse set of anonymization algorithms that FLAIM provides by only creating a module to do the I/O and parsing.¹

Multi-level Anonymization

Security related logs often need to be shared between several different organizations to investigate a compromise, or even internally between different groups within the organization. Logs could also be shared externally with organizations providing security out-sourcing services. The level of anonymization to be applied to the logs changes with the recipient of the log. For internal users, the logs may not have to be anonymized as strongly, whereas for external users they most likely require stronger protections.

This means that an anonymization tool must support multiple levels of anonymization. Different levels could be used for different recipients as well as different situations for log sharing. Having multiple anonymization algorithms of different strengths provides the building blocks for multi-level anonymization, but something more is needed. There should be a system to express and evaluate these anonymization schemes. FLAIM does this through use of XML anonymization policies. Users create these XML policies or use predefined ones to choose the appropriate level of anonymization for a given log. In particular, this policy will specify which fields are anonymized and in what way. Schemas built into FLAIM check that the policies are syntactically correct and that the options make semantic sense for the type of fields. Since these policies are *not* hard coded into FLAIM, they can be easily modified at any time.

Modularity

As we said, FLAIM supports multiple log formats (e.g., netfilter, pcap, nfdump, etc.). We could have done this with a large monolithic piece of software code. In this case, one could use FLAIM for many types of logs, but still only a small set of all the logs out there, namely, the set that we saw as important for our needs. However, we did not do this because we wanted FLAIM to be extensible. We have, instead, created a very modular framework with a strict API between the separate components. Consequently, anonymization, policy verification and parsing have all been separated. This allows not only us, the FLAIM developers, to add support for new types of logs, but it makes it much easier for third party module developers to add support for new types of logs within FLAIM.

FLAIM Architecture

To run FLAIM, one must specify a *user policy* – a description of the anonymization algorithms to be

¹Note: At this time FLAIM modules exist for netfilter logs, pcap traces and nfdump format NetFlows. More modules are under development.

used for each field. One of the key contributions of FLAIM is that it allows the user to specify an anonymization policy that can make use of a multitude of anonymization algorithms at run-time. Another key contribution of FLAIM is that parsing and I/O are separated to allow third parties to add support for additional logs. To achieve this, the architecture is composed of two main components: FLAIM Core and FLAIM Modules. FLAIM Core consists of a suite of anonymization algorithms – the anonymization engine – and the policy manager. FLAIM Core reads and writes to records via the Module API implemented by each I/O module. Figure 1 illustrates the overall architecture of FLAIM.

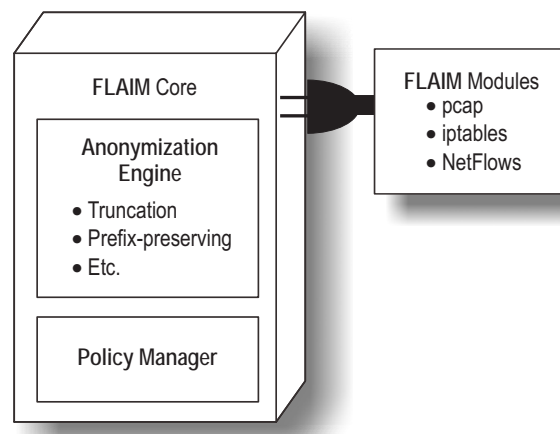


Figure 1: The FLAIM Architecture.

FLAIM modules consist of libraries of methods to parse various types of logs. A single module is normally used to parse a single type of log file. Every module is compiled as a dynamically linked library which loads at runtime. Each module must implement the Module API which defines a set of methods necessary for FLAIM Core to get records to process and return them to the module to be reassembled and written out.

In understanding the design of FLAIM, it is useful to distinguish between three different actors that play a role in the development and use of FLAIM. We have alluded to them before, but define them formally below.

- **LAIM Group:** This refers to the **Log Anonymization and Information Management** group at the NCSA. We developed FLAIM, both FLAIM Core and the first FLAIM Modules.
- **Module Developers:** These people have developed or are interested in developing log parsing modules that implement the Module API.
- **FLAIM Users:** These people are interested in using FLAIM to anonymize a set of logs using existing modules.

The basic workflow for FLAIM is as follows:

1. FLAIM is called with parameters that specify the input/output data, the user policy, and the modules needed to parse the input/output data.

2. The policy manager parses the user policy and determines the anonymization algorithms that will be applied to each field of the data. In addition to parsing, it uses schemas to validate the policy.
3. A record is “requested” by FLAIM Core via the Module API.
4. The record is anonymized based on the user policy.
5. The anonymized record is sent back via the Module API and written out.
6. The last three steps are repeated for all records.

In the rest of this section, we describe the Policy Manager and the Module API. Also, at the heart of FLAIM Core is the Anonymization Engine which is basically a set of anonymization algorithms. The anonymization engine was developed by the LAIM group and is being continually extended. Currently, there are eight different types of anonymization algorithms implemented that differ based on their strength. Section IV describes the various anonymization algorithms in detail, and thus we defer further discussion of the anonymization engine till the next section.

Policy Manager

The policy manager ensures that the anonymization policy specified by the user is valid – that is, it specifies a known anonymization algorithm with valid options for a data type that makes sense. For example, it would not allow prefix-preserving IP address anonymization to be specified for a timestamp field. The anonymization policy must be validated with both the FLAIM schema and the Module schema to be accepted.

The FLAIM schema details what anonymization algorithms are available as well as options for those algorithms. As new versions of FLAIM are released, new anonymization methods will be included in the anonymization engine. The set of anonymization methods currently available is listed in the FLAIM schema. The FLAIM schema is maintained by the FLAIM developers.

The module schema indicates which anonymization methods are appropriate for the specific fields in the log. As the developers of FLAIM, we can specify to what data types a particular anonymization algorithm can be applied. However, we cannot anticipate the names or handles that a module developer will use for those fields. These names are not only used in the anonymization policy, but also in the meta-format that the module uses to send records to FLAIM. In essence, FLAIM Core simply sees a record as a list of tuples of the form $\langle \text{FieldName}, \text{FieldValue} \rangle$. FLAIM Core then matches the field name with the options specified for the field name in the policy to determine how to anonymize it.

We could construct a list of valid field names from which the module developer could choose names. This would allow FLAIM Core to recognize the data type and make sure the algorithm being applied makes semantic sense for that field. However, we could hardly anticipate the needs for any type of log. For example, we could specify “IP1” and “IP2” as valid names, however, a log that we do not anticipate may have four distinct IP addresses per record. It could also have fields of types we did not expect, but need to be passed to the anonymization engine to be kept with the records if they are being reordered.

Consequently, we do not restrict the set of field names that the parser could use when sending records, i.e., the same field names specified in the user’s policy. So to make sure that an algorithm is applied to the correct data type, the module developer creates a schema using a very simple syntax that specifies what algorithms can be used with what field names or handles. Being the only one who knows the data type that corresponds to the field names, this must be done by the module developer. We will distribute header files for the anonymization algorithms so that module developers know the data type expected by an anonymization algorithm.

The module schema is written in XML and must conform to our definition of a “Module Schema

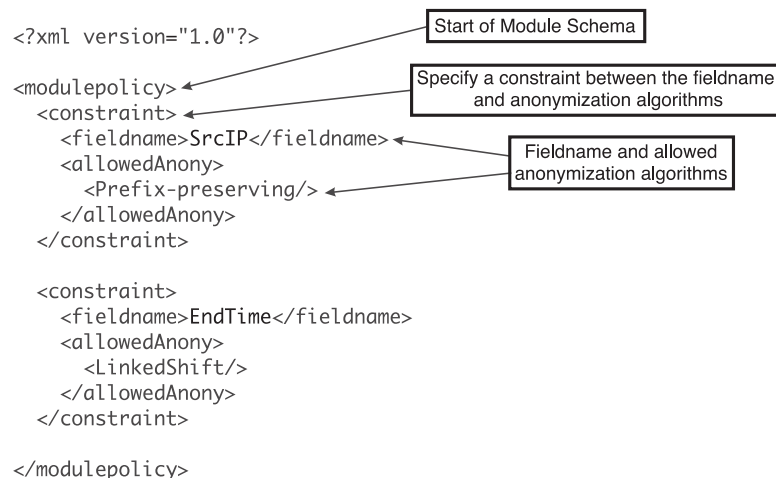


Figure 2: A Module schema written in the *Module Schema Language*.

Language.” For module developers who have experience with Schematron, the module schema may be written in Schematron. We use an XSLT stylesheet to translate the Module schemas written in a simple Module Schema Language to Schematron stylesheets. Figure 2 shows an example module schema written in the Module Schema Language.

We see that, together, validation against both the FLAIM schema and the module schema ensures that the user’s anonymization policy uses anonymization algorithms that are supported by FLAIM, with appropriate options, and are applied to fields in a semantically sound manner (e.g., so an algorithm that only makes sense for timestamps is not applied to IP addresses).

Thus, the policy manager’s role is twofold:

1. Validate the user’s policy against the FLAIM schema and the module schema.
2. Parse the user policy file into an internal format to be used by the anonymization engine.

Figure 3 is an example of part of a sample user policy.

I/O Modules

The I/O library reads and parses an input file or stream. A module library’s purpose is to abstract away from FLAIM the task of physically manipulating the medium which contains the data. All modules must implement the module API that is defined by the LAIM developers. The API allows FLAIM to access the module schema, described above, as well as provides methods to access the data. Modules are responsible for handling storage, retrieval and parsing of the data. Of these data, there are two important types: static files or streams.

Static files are ordinary files to which FLAIM has random access – the key distinction from streams being random access. However, it is not always practical to store all the data in static files. Rather, real-time anonymization may be done on the data as it is collected, perhaps even before it is written to disk. Data that can only be accessed once is called “streaming data,” a particular data source being called a “stream.”

Handling a stream can be quite different from a static file. For example, an anonymization algorithm may require scanning the log twice and hence the ability to go back to the beginning of the data source. This can be problematic for streams, of course. As such, there are mechanisms in the FLAIM API to let the module know if one of the anonymization algorithms chosen will need random access. It is up to the module developer to decide whether or not to support such algorithms and how to do so.

The Module API

The module interface has been designed to be as simple as possible. It consists of five functions that a module must implement. These functions control the input and output of records as well as resetting the counter in the file. Procedures that need to be implemented are listed below:

```
// Returns the filename containing the module schema
char* getModuleSchema()
// Communicate to the module the command
// line parameters passed into FLAIM
void setDataSets(char* inputfilename,
                 char* outputfilename)
// Return a single record
Record getRecord()
// Get the current location of the record counter.
int getCntValue()
// reset the record counter
bool resetCntValue();
// return true if this is the last record
bool atEnd();
// Write a single record
int putRecord(Record r)
```

Summary of the Architecture

The architecture of FLAIM is designed to satisfy the properties stated in the previous section. The Anonymization Engine in FLAIM Core contains a diverse set of anonymization algorithms for many different data types. The Policy Manager component of FLAIM Core allows users to easily provide many different levels of anonymization to logs by utilizing the

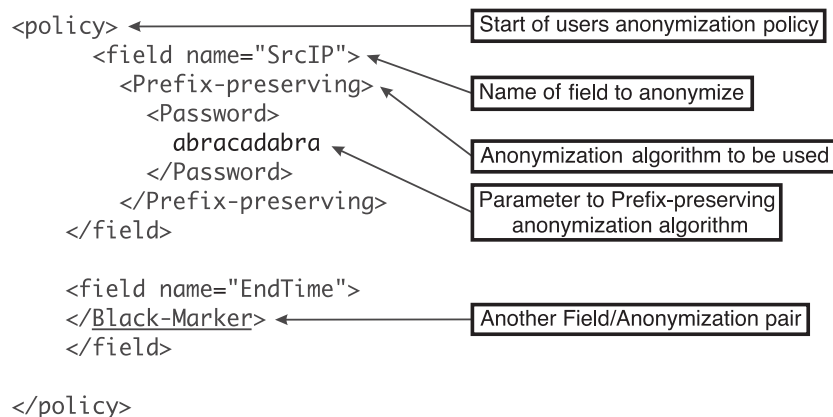


Figure 3: A simple anonymization policy.

many algorithms in the Anonymization Engine. Users can easily change the anonymization of a log by specifying a new user policy in the simple XML language specified by FLAIM. Users can create anonymization policies tailored to the log and situation in which the log is being distributed.

Finally, FLAIM is designed in a modular fashion to be highly extensible to many types of logs. Log parsing modules can be dynamically loaded into FLAIM at runtime. This allows new logs to be added without changing the anonymization engine or the policy manager. We defined a simple API that all log parsing modules must implement. FLAIM Core will interact with the log parsing module via this API. The API is simple, giving module developers much flexibility. Taken as a whole, these components and the modules we have created make FLAIM the first extensible, multi-level, multi-log anonymization tool with the largest set of anonymization primitives of which we are aware.

Anonymization Algorithms for FLAIM

FLAIM implements many types of anonymization algorithms. A few are coupled very tightly with the data type being anonymized, but many can be applied to multiple fields. However, the default values and other configuration options may be affected by the data type. For example, while truncation can be applied to almost any field, it makes sense to truncate a MAC address by 40 bits but not to do the same to a 32 bit IPv4 address. Below we discuss the different classes of anonymization algorithms FLAIM supports, and in the following section we go into an extensive example of how they are used.

Black Marker

Black marker anonymization – a term we coined in [15] – is equivalent, from an information theoretic point of view, to printing out a log and going over each value of a sensitive field with a black marker. This analog variant is often seen in sensitive documents retrieved from the government. We simply implement a digital equivalent.

In our implementation, the entire field or just part of it can be “blacked out.” So IP addresses could have just the last octet “blacked out,” (e.g., 192.168.77.99 could go to 192.168.77.0). In this case, we have replaced the last eight bits of the IP address with 0’s (Though some would call this truncation). However, we did not have to choose to use 0’s. In fact, we allow the user to choose the constant with which to overwrite values. For example, we could anonymize the hostnames by replacing every hostname in a log with *host@example.net*.

Truncation

Truncation works by taking a field and selecting a point after which all bits are annihilated. For a string value, one chooses some middle point – not necessarily defined as a fixed number of characters from the

beginning – and cuts the string off after that point. For example, one could truncate the domain information from e-mail address so that “user@example.net” is replaced with simply “user”.

Truncation will shorten strings, but this is a problem for fixed length binary values. One could simply replace all trailing values with 0’s, but we have called this black marker anonymization. Keeping with the idea that truncating is shortening a value, we pick a point for truncation and right shift until all bits to the right of this point are shifted off the end. For example, consider IP addresses as a binary 32 bit unsigned integer. The IP address 192.168.77.88 could have the last 16 bits truncated which would result in 0.0.192.168. Sometimes, we do not want to work on a number as binary when truncate. Take the decimal number 11977. We could truncate the last two digits, instead of bits, and come up with 119.

Note that truncation and black marker still are not 100 percent mutually exclusive. For example, truncating all 32 bits and replacing all IP’s with the constant address of 0.0.0.0 will have the same result.

Permutation

In the most general sense, a *permutation* is a one-to-one and onto mapping on a set. Thus, even a block cipher is a type of permutation. There are many ways that permutations can be used. For larger binary fields, it usually makes sense to use a strong, cryptographic block cipher. Thus, if one wishes to use the same mapping later, they just save the key. This is excellent for fields like the 128 bit IPv6 address. However, there are no strong 32 bit block ciphers to do the same for IPv4 addresses. Using a larger block would require padding, and the output of the anonymization function would be larger than the input. In these cases, one can use tables to create random permutations. The problem is, of course, that one cannot save these tables as easily as a cryptographic key to keep mappings consistent between logs anonymized at different times.

In addition to random permutations that are cryptographically strong, there is sometimes a need to use structure preserving permutations for certain types of data. When differences between values must be preserved – often the case with timestamps – a simple shift can be used. Shifting all values by a certain number may be an acceptable way to permute values in some instances. For IP addresses, the subnet structure may need to be preserved without knowing the actual subnets (e.g., when analyzing router data). Prefix-preserving pseudonymization is appropriate here. This is a type of permutation uniquely applied to IP addresses, and we discuss it in detail in an example of anonymizing netfilter logs in the next section.

Hash

Cryptographic *hash* functions can be useful for anonymization of both text and binary data. The problem with binary data, of course, is that one must often truncate the result of a hash function to the shorter

length of the field. This avoids breaking log analysis tools that operate on those logs. This also means that the hash function is weaker and has more collisions. For fields 32 bits or smaller, dictionary attacks on hashes become very practical. For example, it is well within the capability of a modern adversary to create a table of hashes for every possible IPv4 address. The space of possible values being hashed is too small. Thus, hash functions must be used carefully and only when the possibility of collision in the mappings is acceptable. Often, it is better to use a random permutation. For string values, FLAIM outputs the hash in an ASCII representation of the hexadecimal value.

HMAC

HMACs (Hash Message Authentication Codes) are essentially hashes seeded with some secret data. Adversaries cannot compute the HMAC themselves without access to the key, and thus they cannot perform the dictionary attacks that they could on simple hashes. In cases where hashes are vulnerable to such attacks, HMACs make sense. However, HMACs, like hashes, are inappropriate when collisions are unacceptable.

Partitioning

Partitioning is just what it sounds like. The set of possible values is partitioned into subsets – possibly by a well-defined equivalence relation – and a canonical example for each subset is chosen. Then, the anonymization function replaces every value with the canonical value from the subset to which it belongs. Black marker anonymization and truncation are really just special cases of partitioning. For example, say that the last eight bits of IPv4 addresses are “blackened out” with 0’s. Then the set of IP addresses is being partitioned into class C networks. Furthermore, the canonical representation is simply the network address of the class C subnet. However, partitioning is not always so simplistic, and our next type of anonymization algorithm is a very unique type of partitioning for timestamps.

Time Unit Annihilation

Time unit annihilation is a special type of partitioning for time and date information. Timestamps can be broken down into year, month, day, hour, minute and second subfields. When this is done, one can annihilate any subset of these time units by replacing them with 0. For instance, if she annihilates the hour, minute and second information, the time has been removed but the date information retained – actually, a type of black marker. If she wipes out the year, month and day, the date information is removed but the time is unaffected. It is clear that this is a very general type of partitioning, but it still cannot partition in arbitrary ways. For instance, it cannot break time up into 10 minute units.

Enumeration

Enumeration can be very general, though FLAIM currently uses it as just an option for timestamps. However, enumeration would work on any

well-ordered set. Enumeration will first sort the records based on this field, choose a value for a first record, and for each successive record, it will choose a greater value. This preserves the order but removes any specific information. When applied to timestamps, it preserves the sequence of events, but it removes information about when they started or how far apart two events are temporally. A straightforward implementation could sort, choose a random starting time, and space all distinct timestamps apart by one second. Note that the output doesn’t actually have to be reordered records. For example, the set {1.2,5.6, 7,0.3,9.3,4.8} could be enumerated with the elements left in place as follows {2,4,5,1,6,3}. However, sorting on the anonymized and unanonymized field values would produce the same result as order is preserved by enumeration.

Netfilter Anonymization: An Extended Example

In this section, we look at how we have applied the various anonymization algorithms described above to netfilter logs – the first type of log we supported with FLAIM. Since all of these fields are also found in pcap traces, we can apply these same algorithms to pcap logs. We describe which algorithms make sense for each field and how they are uniquely adapted to each data type (e.g., how default values change).

Time Stamps

FLAIM supports three methods of timestamp anonymization. As with any data type, we must choose a canonical form for the field. We are using the traditional UNIX epoch format, i.e., the number of seconds since Jan 1st, 1970. This is fastest for two of the three applicable anonymization algorithms. Because timestamps often occur in pairs, with a starting and ending time, we optionally allow a handle to a secondary timestamp field to be specified in the anonymization policy. If the secondary timestamp is specified in the anonymization options, it is adjusted with the first timestamp so that the difference between the two values remains the same. For example, this could be used to keep flow duration the same in NetFlows.

Time Unit Annihilation

FLAIM can use time unit annihilation, as described in the previous section, to anonymize timestamps. Timestamps are converted in the anonymization engine from the canonical format to an internal structure to perform this type of anonymization. Then they are converted back.

Random Shift

In some situations it may be important to know how far apart two events are temporally without knowing exactly when they happened. For this reason, a log or set of logs can be anonymized at once such that all timestamps are shifted by the same random number, in seconds. We call this method of anonymization a *Random Time Shift*. As noted in the previous section, this is a special type of permutation.

We allow a lower and upper bound for the random number to be set for two reasons. First, an upper bound can prevent a shift so far into the future that it overflows the 32 bit timestamp field and events wrap-around back to Jan. 1st, 1970. Second, by setting the lower and upper bound equal, you can control exactly how much the time stamps shift. This allows you to keep the anonymization mapping consistent between different runs of FLAIM.

Enumeration

We implement the enumeration method of anonymization described in general terms earlier in this section. FLAIM chooses a random starting time for the first record, and each subsequent timestamp – if differing from the previous – is one second later. The end result is that one can tell if two records happened at the same time or one before the other, but they can know nothing else. Implementing the enumeration algorithm exactly as described does pose one problem, however. The presorting can be slow on even medium sized logs and nearly impossible with streamed data. Though logs are not always presorted, they are often close to being in order. Often they are simply out of order because of clock skew between different data sources.

In our implementation, we exploit the fact that timestamps are often just slightly out of order (e.g., NetFlows are nearly in order by ending timestamp) and buffer events to sort locally. This buffer is used like a sliding window in which only events within the window can be sorted. Events before the window are written out already, and events after window have not yet been read. Since events are usually not terribly disordered, this sorts records with great accuracy often when using a small window. The size of the window is user-selectable. Larger windows must be used for logs that are more disordered. We have found this to be a useful way to allow users to select a level of compromise between efficiency and accuracy.

IP Addresses

We have implemented four types of IP address anonymization in FLAIM: truncation, black marker, prefix-preserving, and random permutation. Nearly all of the log anonymization tools out there use one of these algorithms or hashing – which can be easily brute-forced on the small 32 bit space for IPv4 addresses. Others are slight variations, such as permutations that fix a hard-coded set of internal IP addresses. In FLAIM, the canonical form for an IPv4 address is a 32 bit unsigned integer.

Truncation and Black Marker

FLAIM will truncate IP addresses from 1 to 32 least significant bits. Similarly, one can black out any number of least significant bits and replace those bits with any constant.

Random Permutation

We also support anonymization by creating random permutations on the set of possible IP addresses.

This permutation is then applied to each IP address in the log. We implement this algorithm through use of two hash tables for efficient lookup. One hash table is used to store mappings from unanonymized to anonymized IP addresses. The other hash table is used to store all of the anonymized addresses so we can check if an address has already been used when creating a new mapping. Because the first table is indexed by unanonymized addresses, the whole table would have to be searched for a free anonymized address if we did not use a second hash table. In this way, we trade a little storage for a large computational speed-up. For even higher space efficiency we could use a Bloom filter [3] to store the set of used IP addresses. Since Bloom filters never give a false negative, we would not map two distinct IP addresses to the same value, and thus our function would remain injective.

Sometimes it is desirable to fix certain elements within the permutation. Say that an organization wants all external IP addresses to remain unchanged. This sort of less random permutation can be implemented by simply pre-filling the tables with entries that fix this subset of elements. Future versions of FLAIM may allow one to specify a CIDR addressed subnet to fix in the permutation.

Prefix-preserving Permutation

Prefix-preserving pseudonymization uses a special type of permutation that has a unique structure preserving property. The property is that two anonymized IP addresses match on a prefix of n bits if and only if the unanonymized addresses match on n bits. This preserves subnet structures and is often preferred to random permutations, but the simple fact that there are many times fewer permutations of this type makes it weaker.

As a general principle, cryptographic algorithms that preserve structure are more open to attack, and this algorithm is not an exception. For example, an adversary that injects traffic to be recognized later not only gleams information about the addresses she specifically attacked, but she also learns many of the unanonymized bits of addresses that share prefixes with the addresses she attacks. For this type of anonymization we implemented the Crypto-PAn [20, 21] algorithm and generate keys by hashing a passphrase the user provides. In this way, tables are not used, and logs can more easily be anonymized in parallel across different locations. This is a sensible choice when an injection attack is unlikely to be effective (e.g., a one-time release of logs with a particular key) and the subnet structure is very important.

MAC Addresses

MAC addresses have traditionally been globally unique, and the first three of six bytes are usually indicative of the network card manufacturer. As such they are somewhat sensitive. However, proliferation of virtualization means that MAC addresses are not

always globally unique now. In addition, many hardware devices allow you to change the MAC (e.g., SOHO routers). FLAIM supports three types of anonymization for MAC addresses and uses a six byte unsigned char array as the canonical representation.

Truncation and Black Marker

FLAIM will truncate or black out any number of least significant bits. This could allow one to remove all identifying information but the manufacturer or allow one to obscure the entire address.

Random Permutation

It may be important to distinguish network interfaces within a log, but not to know the specific MAC. In fact, this is often the case since knowing the specific MAC usually does not get you any information except for the manufacturer unless you have access to special outside knowledge (e.g., access to ARP or DHCP logs). For this reason we have an algorithm that creates a random permutation of MAC addresses. It is implemented in the same manner as the random permutation of IP addresses.

Hostnames

FLAIM can also anonymize hostnames that are both local and fully qualified with domain names. The canonical form of a hostname is a string. If there are periods in the hostname, the host part is to the left of the first period and the domain name is the part to the right of the first period. Whether or not the hostname is fully qualified, can make a difference in the anonymization function.

Black Marker

Black marker anonymization replaces fields with constants. In the context of hostnames, FLAIM can be configured to black marker anonymize just the host part or the entire name. If it is configured to black marker anonymize just the host part, the host part of the name is replaced with the string "host". If the hostname is fully qualified, the domain name is left untouched. For example, a hostname of "vorlon" would be replaced with "host". A hostname of "vorlon.ncsa.uiuc.edu" would be replaced with "host.ncsa.uiuc.edu".

Black marker anonymization can also be configured to replace the entire hostname. If the hostname is fully qualified, it is replaced with "host.network.net", otherwise it is simply replaced with "host". In addition to setting the black marker anonymization to the host or full name, one can specify the constant strings with which to replace the names.

Hash

Another anonymization algorithm available for hostnames is a simple hash converted into ASCII output. While we could hash just the host part and leave the network part alone, this is not very useful since the valid hostnames on a given network can be easily enumerated in most cases, and thus the hash function could be brute-forced by a simple dictionary attack. Therefore, we only hash the entire string.

HMACs

As noted earlier in this section, HMACs can be used instead of hashes when there is concern of brute-force attacks. Future revisions of FLAIM will add support for using HMACs to anonymize hostnames.

Port Numbers

We have implemented three methods of anonymization for port numbers in FLAIM. The canonical representation for port numbers is a 16 bit unsigned integer.

Black Marker

FLAIM supports black marker anonymization of this field, replacing every port number with port 0.

Bilateral Classification

We call the second method of port number anonymization we implement *bilateral classification*. This is really just a special type of partitioning. Often the port number is useless unless one knows the exact port number to correlate with a service. However, there is one important piece of information that does not require one to know the actual port number: whether or not the port is ephemeral. In this way ports can be classified as being below 1024 or greater than or equal to 1024. The canonical value for privileged ports is 0, and the canonical value for ephemeral ports is 65535.

Random Permutation

In some cases it may not be necessary to know what the port is, but only that some ports are seeing particular traffic patterns. Such is the case in detecting worms and P2P traffic. Such traffic has unique characteristics, and knowing the exact port number may not even be that useful. For example, a new worm may appear on an unexpected port. If you are just counting on the port number, you will not recognize this new worm. However, if you look for worm-like behaviors appearing on a fixed port, you can detect the worm [22]. Similarly, knowing the specific port of a piece of P2P software is becoming less useful, as the ports are becoming dynamic. On a particular machine the port will be fixed for a while, but that particular machine often chooses a random port. Thus, detection must depend on elements other than knowing the specific port number.

Randomly permuting the port numbers does not affect the ability to detect worms or P2P traffic using these more advanced methodologies since such services do not always use the same predictable port numbers. However, bilateral classification or black marker anonymization may affect such methodologies as it is no longer easy to separate the traffic from different applications on the same machine. For example, a machine may have a P2P application on port 7777 and a web server on port 8080. Depending on the log and the specific behavior of the P2P application, it may be difficult to detect the P2P application as all of its traffic is aggregated with the legitimate web server. In this situation, a random permutation provides the maximum amount of anonymization possible to complete the analysis.

Network Protocol

FLAIM supports black marker anonymization of the protocol field. The canonical form of this field is an unsigned eight bit integer corresponding to the values assigned by IANA [6] (e.g., TCP is 6, UDP is 17, ICMP is 1). If this field is anonymized, all protocols are replaced with 255, the IANA reserved protocol number. For many network logs it makes no sense to anonymize this field alone. For example, in a pcap log the TCP headers will give away the protocol even if the protocol field is eliminated in the IP header.

IP ID Number

We support black marker anonymization of ID numbers because their use is moderately sensitive to passive OS fingerprinting. All ID's are replaced with 0 in this case. The canonical form of this field is an unsigned 16 bit integer.

IP Options

We support black marker anonymization of IP options because their use is moderately sensitive to passive OS fingerprinting and because they can carry significant information in covert channels. Specifically, we can see their usefulness in data injection and probing attacks. The canonical form for this field is a string in the same format that iptables uses to represent this variable length field. If black marker anonymization is chosen, FLAIM replaces the string with the null string.

Misc IP Fields

We support black marker anonymization of Type-of-Service and Time-to-Live IP fields because their use is very sensitive to passive OS fingerprinting. Their canonical form is the same as the protocol field, and we replace all values with 255 when performing this type of anonymization.

Don't Fragment Bit

We support black marker anonymization of Don't-Fragment bits because this field is very sensitive to passive OS fingerprinting. Nothing smaller than a byte is sent to FLAIM, so any flags or boolean values have the canonical form of an unsigned char. If the value is non-zero, it is treated as if the bit is set to true. If the value is zero, it is interpreted that bit is not set or set to false. Thus, when FLAIM anonymizes this field, it simply replaces instances of the field with 0.

TCP Window Size

We support black marker anonymization of the TCP window size because its use is very susceptible to passive OS fingerprinting. The canonical form of this field is a 16 bit unsigned integer. If anonymization of this field is done, all window sizes are set to 0.

Initial TCP Sequence Number

The canonical form of this field is an unsigned 16 bit integer. Because initial TCP sequence numbers are moderately sensitive to passive OS fingerprinting, FLAIM

can be used to black marker anonymize TCP sequence numbers, replacing all sequence numbers with 0.

While doing this is not syntactically breaking logs, it will lead to semantically nonsensical values. Log analyzers should not break *while parsing* logs anonymized in this way, but their output could be quite unpredictable when fields like this are anonymized in such a manner. More complex anonymization algorithms that try to identify when packets are part of the same flow could be used, instead. Their drawback is that the anonymization of one field becomes dependent on other fields within a record. FLAIM is certainly general enough to do this, and we do in fact handle relations between timestamp fields within a record. However, for a field that has seen lesser demand for anonymization, we have chosen to perform a simpler type of anonymization in this first instance of FLAIM. User demand will determine if more complex solutions are desired for some of these fields.

TCP Options

We support black marker anonymization of TCP options because their use is moderately sensitive to passive OS fingerprinting and because they can carry significant information in covert channels. Specifically, we can see their usefulness in data injection and probing attacks. The canonical form for this field is a string in the same format that iptables uses to represent this variable length field. If black marker anonymization is chosen, FLAIM replaces the string with the null string.

ICMP Codes and Types

NetFlow records and firewall logs can indicate both the ICMP code and type without the packet data. Truncation makes no sense since there is no structure to the specific code and type numbers. Permutations and hashing seem to offer no utility over simple black marker anonymization. Thus, FLAIM currently supports just black marker anonymization of these fields. The canonical form for these fields is the unsigned char. The value with which these fields are replaced is 0.

Figure 4 summarizes the anonymization methods that can be applied to the fields in the IP table logs.

Related Work

While there have been several anonymization tools created for specific logs, they have not been very flexible to date. The anonymization primitives used have mostly been simplistic, and anonymization is typically done on only one field with no options as to what anonymization algorithm is used. Thus, in the current state of matters, we have a collection of ad hoc tools created for the specific needs of individual organizations, rather than flexible tools used by many.

One of the major results in log anonymization – one that changed how a lot of tools anonymize data – was the development of prefix-preserving IP address

anonymization. The most commonly anonymized field in security and network relevant logs is the IP address. It was long desired to both preserve the subnet structure like truncation, and also have a one-to-one mapping between anonymized and unanonymized addresses. The solution was to create prefix-preserving permutations on IP addresses. Mathematically, we define such a mapping as follows. Let τ be a permutation on the set of IP addresses, and let $P_n()$ be the function that truncates an IP address to n bits. Then τ is a *prefix-preserving* permutation of IP addresses if $\forall 1 \leq n \leq 32$:

$$P_n(x) = P_n(y) \text{ if and only if } P_n(\tau(x)) = P_n(\tau(y)).$$

In recent years, several tools have been made that make use of this newer, structure preserving form of IP address anonymization. Many are based on `tcpdpriv`, a free program that performs prefix-preserving `pcap` trace anonymization using tables. Because of the use of tables, it is difficult to process logs in parallel with this tool. In [20, 21], Xu, et al., created a prefix-preserving IP pseudonymizer that overcomes this limitation by eliminating the need for centralized tables to be shared and edited by multiple entities. Instead, with their algorithm `Crypto-PAn`, one only needs to distribute a short key between entities that wish to pseudonymize consistently with each other. Their work made prefix-preserving pseudonymization much more practical. In [14], we used `Crypto-PAn` with our own key generator to perform prefix-preserving IP address pseudonymization on a particular format of NetFlow logs. We later implemented `Crypto-PAn` in Java as part of a more advanced NetFlow anonymizer we call `CANINE` [17]. `CANINE` supports several NetFlow formats and anonymizes the eight most common fields within NetFlows.

In [18], Sobirey, et al., first suggested privacy-enhanced intrusion detection using pseudonyms and provided the motivation for the work of Biskup, et al., in [1, 2]. While the work in [9, 1, 2] does deal with log data and anonymization, their goals are significantly different than ours. All three works deal specifically with pseudonymization in Intrusion Detection Systems (IDSs). The adversary in their model is the system administrator, and the one requiring protection is the user of the system. In our case, we instead assume that the system/network administrators have access to raw logs, and we are trying to protect the systems from those who would see the shared logs. To contrast how this makes a difference, consider that in their scenario the server addresses and services running are not even sensitive – just information that could identify clients of the system. Furthermore, we do not require the ability to reverse pseudonyms.

However, since the system/network administrators in their scenario do not have raw data, the privacy officer must help the system security officer reverse pseudonyms if alerts indicate suspicious behavior. In [1, 2], they take this further and try to support automatic re-identification if a certain threshold of events is met. In that way, their pseudonymizer must be intelligent, like an IDS, predicting when re-identification may be necessary and thus altering how it pseudonymizes data. They also differ from us in that they create transactional pseudonyms, so a pseudonym this week might map to a different entity the next week. We, however, desire consistency with respect to time for logs to be useful. Lastly, all of the anonymizing solutions in these papers filter log entries and remove them if they are not relevant to the IDS; we endeavor to dispose of no entries because completeness is very

IP Address	Port Name	Don't Fragment Bit
- Truncation	- Black Marker	- Black Marker
- Black Marker	- Bilateral	
- Prefix-Preserving	- Random Permutation	
- Random Permutation		
Time Stamp	Network Protocol	TCP Window Size
- Time Unit Annihilation	- Black Marker	- Black Marker
- Random Shift		
- Enumeration		
MAC Address	IP ID	Initial TCP Sequence Number
- Truncation	- Black Marker	- Black Marker
- Black Marker		
- Random Permutation	IP Options	TCP Options
	- Black Marker	- Black Marker
Host Name	ICMP Codes and Types	Misc IP Fields
- Black Marker	- Black Marker	- Black Marker
- Hash		
- HMAC		

Figure 4: NetFilter Anonymization Options with FLAIM.

important for logs released to the general research populace. All-in-all, we are looking at the more general problem of sharing arbitrary logs, rather than hooking anonymization into IDS's or other tools for very specific purposes.

In [4], Flegel takes his previous work in privacy preserving intrusion detection [1, 2] and changes the motivation slightly. Here, he imagines a scenario of web servers volunteering to protect the privacy of visitors from themselves, and he believes IP addresses of visitors need pseudonymization. However, to a web server IP addresses already act as a pseudonym protecting the client's identity, since ISPs rarely volunteer IP-to-person mappings to non-government entities. Though the motivation differs slightly, the system described is the same underlying threshold based pseudonymization system, and the focus of this paper is really about the implementation and performance of the system. As such, the results of [4] can be applied to [1, 2].

In [10], Pang, et al., developed a new packet anonymizer that anonymizes packet payloads as well as transactional information, though their methodology only works with application level protocols that their anonymizer understands: HTTP, FTP, Finger, Ident and SMTP. The process can also alter logs significantly, losing fragmentation information, the size and number of packets and information about retransmissions; thus skewing timestamps, sequence numbers and checksums. While their anonymizer is limited in its capabilities, it is fail-safe because it only leaves information in the packets that it can parse and understand. Further, they create a classification of anonymization techniques and a classification of attacks against anonymization schemes that we found useful. We use a similar classification which is based off of their work.

Lincoln, et al., [8] proposed a log repository framework that enables community alert aggregation and correlation, while maintaining privacy for alert contributors. However, the anonymization scheme in the paper is partially based on hashing IP addresses. Such a scheme is always vulnerable to dictionary attacks. Moreover, their scheme mixes the hashes with HMACs and truncates the hashes/MACs to 32 bits, both actions which result in more hash collisions and inconsistent mappings. Finally, their suggested use of re-keying by the repository destroys the correlation between repositories and therefore limits the view to a single repository.

Most recently, Pang, et al., have taken a new approach to packet trace anonymization, ignoring packet data [11]. Their new tool anonymizes many fields in pcap logs, more than any other tool to date. The algorithm options to anonymize these fields are very customized to their specific needs which they describe through the paper much like a case study. However, they leave hooks into the software that would allow someone to create new algorithms to

anonymize any TCP/IP fields in new ways. The difference between our work is that theirs is (1) restricted to pcap headers, (2) is not a modular framework with a clean API, and (3) not many options are available for any particular field, though the ability for someone else to code another algorithm is there. Another way to look at the difference is that they just look at pcap headers and leave open the ability for others to add anonymization algorithms to their parser. We create a suite of anonymization algorithms with a policy manager, and recognizing that many logs share common fields, we leave open the ability to add new parsers.

Software Availability

FLAIM software is available at <http://flaim.ncsa.uiuc.edu> for download. This software is released open source under the University of Illinois open source license. The license and contact information can also be found on the FLAIM web site.

Conclusions and Future Work

As we can see, the ability to share logs can vastly improve the detection of zero-day exploits and other network attacks. In addition, the sharing of network traces will allow multiple organizations to pursue research on real-life examples, instead of simulations. The main problem with sharing logs is to make sure that sensitive data is not compromised by sharing the logs. But while we should make sure sensitive information is not distributed, we must also make sure enough information is retained for analysis. We propose that anonymization can be used to sanitize the sensitive information in a log while keeping enough information for analysis. Our goal is to explore the trade-offs between the strength of an anonymization algorithm (the amount of information it hides) and its utility (the usefulness of the log after anonymization). We have developed a tool, the Framework for Log Anonymization and Information Management (FLAIM), which we will use to explore these issues.

Log anonymization tools to date have been created in an ad hoc manner for the specific needs of individual organizations. FLAIM is immanently more flexible than the log anonymization tools to date. FLAIM is extensible, supports multi-level anonymization with a rich supply of anonymization algorithms and supports multiple log formats. We have discussed its goals and architecture in depth, and found it to perform well at anonymizing large log files (over 40,000 records per second²). With the release of FLAIM, organizations are now able to leverage our work to address their unique log sharing needs.

FLAIM was developed with the purpose of aiding security engineers in safely sharing security related data with other professionals. While this is still the

²Tests were performed on a machine with four 3.0 GHz Xeon CPUs and 2 GB of RAM against an nfdump log with 5 million flows.

main goal of FLAIM, it is important to note that FLAIM can be used on any type of data, provided a module is written for it. We have seen that there is a need for anonymization to allow the further progress of science. Because of FLAIM's separation of the file I/O from the actual anonymization, FLAIM can provide a general framework for anonymization of any data. Any developer can make use of the diverse set of anonymization algorithms we provide to anonymize their data. By releasing FLAIM as an open source tool, and by creating a simple module API, we hope that many professionals will build modules for FLAIM.

While FLAIM is a capable tool, there are always enhancements that can be made. New parsing modules may be written for parsing additional log types (We intend to create new modules for more types of system logs and the IDMEF IDS format). These parsing modules can also be enhanced to support streaming or real-time anonymization. We also foresee interest in a daemon mode for FLAIM that would make it easier to integrate with web services and other online tools. Most interesting to us, is the creation of a tool to help generate anonymization policies. It is always challenging to determine how best to anonymize the data for a particular application. We envision a tool that asks users a series of questions to determine how to anonymize a log, and then it would output a valid XML policy for FLAIM. As we receive feedback from users, we expect to have a more solid grasp of which features are most useful.

Author Biographies

Adam Slagell received a B.S. and M.S. in mathematics at Northern Illinois University. Afterwards, he received an M.S. in computer science at the University of Illinois at Urbana-Champaign in 2003. After obtaining his second masters degree, he joined the security research group at the National Center for Supercomputing Applications where he is still employed. He can be reached electronically at slagell@ncsa.uiuc.edu.

Kiran Lakkaraju is a Ph.D. student in the Computer Science Department at the University of Illinois at Urbana-Champaign. He has been working for the National Center for Supercomputing Applications for four years on projects involving security visualization and anonymization. His other research interests include multi-agent systems and language evolution. He can be reached electronically at kiran@ncsa.uiuc.edu.

Katherine (Xiaolin) Luo is currently studying for her masters degree in the general engineering department at the University of Illinois at Urbana-Champaign. She is also working as a graduate research assistant for the LAIM group at the NCSA, and is one of the main developers of the FLAIM tool.

Acknowledgments

This material is based, in part, upon work supported by the National Science Foundation under

Grant No. 0524643, and the Office of Naval Research through the National Center for Advanced Secure Systems Research (NCASSR). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Office of Naval Research.

We would like to thank Greg Colombo, Vikram Dhar and Yifan Li who have contributed their programming skills to the development of FLAIM. We would also like to thank Jun Wang for his help reviewing this paper before submission, and Bill Yurcik for help in various ways.

Bibliography

- [1] Biskup, J., and Flegel, U., "On Pseudonymization of Audit Data for Intrusion Detection," *USENIX Workshop on Design Issues in Anonymity and Unobservability*, Jul., 2000.
- [2] Biskup, J., and Flegel, U., "Transaction-Based Pseudonyms in Audit Data for Privacy Respecting Intrusion Detection," *Third International Workshop on the Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, Oct., 2000.
- [3] Bloom, B. H., "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, Vol. 13, Num. 7, pp. 422-426, 1970.
- [4] Flegel, U., "Pseudonymizing UNIX Log Files," *Infrastructure Security, International Conference (InfraSec 2002)*, Bristol, UK, Oct, 2002.
- [5] Gorman, S., "NSA Killed System that Sifted Phone Data Legally," *The Baltimore Sun*, May 17, 2006.
- [6] IANA IPv4 Protocol Numbers Assignment, <http://www.iana.org/assignments/protocol-numbers>, Mar., 2006.
- [7] Markoff, J., and Bergman, L., "Internet Attack is called Broad and Long Lasting," *New York Times*, Sec. A, Col. 1, p. 1, May 10, 2005.
- [8] Lincoln, P., Porras, P., and Shmatikov, V., "Privacy-Preserving Sharing and Correlation of Security Alerts," *13th USENIX Security Symposium*, San Diego, CA, Aug., 2004.
- [9] Lundin, E., and Jonsson, E., "Privacy vs Intrusion Detection Analysis," *Second International Workshop on the Recent Advances in Intrusion Detection (RAID '99)*, West Lafayette, IN, Sep., 1999.
- [10] Pang, R., and Paxson, V., "A High-Level Programming Environment for Packet Trace Anonymization and Transformation," *ACM SIGCOMM Conference*, Karlsruhe, Germany, Aug., 2003.
- [11] Pang, R., Allman, M., Paxson, V., and Lee, J., "The Devil and Packet Trace Anonymization," *ACM SIGCOMM Computer Communications Review*, Vol. 36, Num. 1, pp. 29-38, Jan., 2006.

- [12] Poulsen, K., "California Reports Massive Data Breach," *SecurityFocus News*, <http://www.securityfocus.com/>, October 19, 2004.
- [13] Shannon, C., Moore, D., and Keys, K., "The Internet Measurement Data Catalog," *ACM SIGCOMM Computer Communications Review*, Vol. 35, Num. 5, pp. 97-100, Oct., 2005.
- [14] Slagell, A., Wang, J., and Yurcik, W., "Network Log Anonymization: Application of Crypto-PAN to Cisco NetFlows," *Workshop on Secure Knowledge Management*, Buffalo, NY, Sep., 2004.
- [15] Slagell, A., and Yurcik, W., "Sharing Computer and Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization," *ACM Computing research Repository (CoRR)*, Technical Report cs.CR/0409005; Sep., 2004.
- [16] Slagell, A., and Yurcik, W., "Sharing Computer Network Logs for Security and Privacy: A Motivation for New Methodologies of Anonymization," *SECOVAL: The Workshop on the Value of Security through Collaboration*, Athens, Greece, Sep., 2005.
- [17] Slagell, A., Li, Y., and Luo, K., "Sharing Network Logs for Computer Forensics: A New tool for the Anonymization of NetFlow Records," *Computer Network Forensics Research Workshop*, Athens, Greece, Sep., 2005.
- [18] Sobirey, M., Fischer-Hubner, S., and Rannenburg, K., "Pseudonymous Audit for Privacy Enhanced Intrusion Detection," *IFIP TC11 13th International Conference on Information Security*, Copenhagen, Denmark, May, 1997.
- [19] Vrabie, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker, G., and Savage, S., "Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm," *20th ACM Symposium on Operating Systems Principles (SOSP 2005)*, Brighton, UK, Oct., 2005.
- [20] Xu, J., Fan, J., Ammar, M. H., and Moon, S. B., "On the Design and Performance of Prefix-Preserving IP Traffic Trace Anonymization," *ACM SIGCOMM Internet Measurement Workshop*, San Francisco, CA, Nov., 2001.
- [21] Xu, J., Fan, J., Ammar, M. H., and Moon, S. B., "Prefix-Preserving IP Address Anonymization: Measurement-based Security Evaluation and a New Cryptography-based Scheme," *10th IEEE International Conference on Network Protocols*, Paris, France, Nov., 2002.
- [22] Yin, X., Yurcik, W., and Slagell, A., "The Design of VisFlowConnect-IP: a Link Analysis System for IP Security Situational Awareness," *First International Workshop on Information Assurance (IWIA)*, College Park, MD, Mar., 2005.

Centralized Security Policy Support for Virtual Machine

Nguyen Anh Quynh, Ruo Ando, and Yoshiyasu Takefuji – Keio University

ABSTRACT

For decades, researchers have pointed out that Mandatory Access Control (MAC) is an effective method to protect computer systems from being misused. Unfortunately, MAC is still not widely deployed because of its complexity. The problem is even worse in a virtual machine environment, because the current architecture is not designed to support MAC in a site-wide manner: machines with multiple virtual hosts need to have multiple MAC security policies, and each of these policies must be updated and managed separately inside each virtual host.

In order to ease the burden on administrators when deploying security policies in a virtual environment, this paper proposes an architecture named *Virtual Mandatory Access Control (VMAC)* to centralize security policies, so that all policy management can easily be done from a central machine. VMAC securely centralizes the security logging information from all virtual hosts into a central machine so intrusion detection analysis on the logging data is straightforward.

To arrive at the architecture presented here, we have investigated various popular MAC schemes, and implemented several schemes with VMAC on the Xen Virtual Machine. This paper presents our experiences in the development process.

Introduction

In today's world, in which almost everything relies on a computer to work, information assurance becomes very important. Computer security is primarily done through the enforcement of security policy, which sets a context within which the notion of a secure system can be defined. If we consider a system to be a finite-state automaton with a set of states, a security policy is a statement that partitions the states into secure and non-secure states.

Basically, we can classify security policies into two classes. The first class, *discretionary security policy*, consists of any security policy which ordinary users can define, alter or assign. The second class, *mandatory security policy* is different: this kind of policy must be defined and controlled tightly by the system administrators, enforced at the operating system level.

Correspondingly, a security policy may use two types of access control: *Discretionary Access Control* (or *DAC* in short) or *Mandatory Access Control* (or *MAC* in short). It is also possible to combine them together, as usually seen in current practice.

- With DAC, a user can set an access control mechanism to allow or deny others access to a system object he owns. DAC bases access rights on the identity of the subject and the identity of the object involved, in which the owner of the object can specify which subjects can access the object, with particular rights.
- In contrast with DAC, MAC does not rely on the identity of the subject: when the access

control is set on an object (typically by the system administrator), the owner of that object cannot modify the access right. In this case, the OS is responsible for enforcing the MAC, and inspects information associated with both the subject and object to determine whether to give the access right to the subject.

In the real world, DAC is widely used in all modern operating systems because it is very simple, easy to use, and easy to manage. However, DAC is the root of most security issues today. The reason is that with DAC, it is impossible to protect the system against malicious code (such as hostile mobile code or Trojan horses) because such code gains the security permissions of the user executing them. Exploiting these legitimate rights, the malware gains unauthorized access to the user's files and resources, and can then compromise their confidentiality and integrity. Even worse, it can change the DAC policy to abuse the system without the user's knowledge. This causes an arms race between anti-malware researchers and the bad guys, and the problem becomes more and more serious as time goes on [1, 2].

To address the problems of DAC, it is necessary to eliminate them and employ MAC instead [3]. In fact, for decades various researchers have demonstrated that mandatory security is essential for security of computer systems, so MAC should be used to enforce security policy instead of DAC [4]. With MAC, a malicious user (or malicious code running on behalf of a legitimate user) cannot misuse resources for a purpose other than what the system specifies. Because MAC can only be changed by the system

administrator, harmful code cannot modify the policy without approval from the system.

To realize the MAC capability, various schemes and implementations have been introduced in all kinds of operating systems, with the hope that all systems would eventually switch from DAC to MAC. Unfortunately, while MAC might efficiently stop or mitigate security damages, many proposed schemes proved hard to deploy and complicated to manage. MAC often becomes so painful that administrators choose to disable MAC and return to DAC instead, despite the fact that MAC would help to secure their machines [5]. Currently, researchers are working on making MAC easier to use and more widely accepted.

Recently virtual machine (VM) technology has emerged as one of the hottest topics in computer research. The principle of VM technology is to allow the creation of many virtual hosts, each running an instance of an operating system, all running at the same time on the same physical machine. Obviously VM technology can help to reduce both hardware and maintenance costs for organizations that need to use various machines for different services.

However, systems with virtual hosts face a major problem when deploying MAC: current schemes do not address MAC within a VM environment. A machine with multiple virtual hosts needs to have multiple security policies, and each of these policies must be managed separately inside each virtual host. Whenever the administrator wants to update the security policy of a specific host, he must login to that host and carries out the job there. As a result, managing machines with hundreds or more virtual hosts is very time-consuming and complicated, especially if the hosts employ different policies. Clearly, we need more efficient and flexible methods to solve this problem.

In an attempt to address this difficulty, this paper proposes a novel architecture named Virtual Mandatory Access Control (*VMAC*) for virtual hosts: instead of having security policies inside of each host, VMAC moves the security policies outside of the VMs, and puts them into a central machine. This strategy allows the administrator to manage the security policies of all the virtual hosts from an administrative virtual host. As a result, updating the MAC for virtual hosts becomes much easier, because it can be done from a central place. The architecture also gathers the security logging data from virtual hosts and saves them into the central machine, so the intrusion detection process based on security violation logging can also be done there. Obviously, this scheme makes it easier to correlate and manage intrusion evidence. To prove the concept of VMAC, we have implemented it on the Xen Virtual Machine [6, 7, 8], with three selected MAC schemes: AppArmor, LIDS and Trustees.

The rest of this paper comprises five sections: a summary of related works, the VMAC architecture,

and implementation of VMAC on Xen, experiences employing selected MAC schemes, a few concerns with the proposed solution, and a conclusion with future plans.

Related Works

Our work is a combination of two ideas: MAC and operating system virtualization. The original meaning of an operating system virtual machine, also called a hardware virtual machine, is that of a number of different identical execution environments executing on a single computer. One of the most popular uses of virtual machines is to allow a user to run multiple full-featured operating systems at the same time on one physical machine. The host software providing this ability is often referred to as a virtual machine monitor (VMM) or hypervisor.

Recently, virtualization has emerged as one of the hottest research topics, garnering much attention from academia and industrial sectors. Various VM software systems are available, ranging from commercial solutions such as VMWare or Microsoft Virtual PC to free solutions like UML, OpenVZ, VServer and Xen.

Among these technologies, the Xen Virtual Machine Monitor is one of the most interesting solutions. Basically, Xen is a thin layer of software above the bare hardware. Xen exposes a VM abstraction that is only slightly different from the underlying hardware. Xen introduces a new architecture called *xen*, which is very similar to the x86 architecture. The VMs executing on Xen are modified (at the kernel level) to work with the *xen* architecture. With this Xen-aware kernel, the performance impact of the virtual hosts is amazingly low: only around 3% in some published experiments [9]. All accesses of virtual hosts to the hardware and peripherals must go through Xen, so that Xen can keep a close eye on those VMs and control all activities.

Formerly developed by researchers from Cambridge University, currently Xen is backed by various industrial players such as IBM, Intel, AMD, HP, Red-Hat and Novell. The whole Xen community is working very hard to push the code into the Linux kernel, so that it will be available for everyone to use.

Mandatory security has been a research topic for decades in the research community. Originally, the Orange Book defined mandatory security as the multi-level security policy of the Department of Defense [10]. However, in this paper we use a more general notion of mandatory security, in which we consider a security policy mandatory if the definition of the policy and the assignment of security attributes are tightly controlled by the system administrator. This is the most common use of this terminology nowadays [4].

To realize mandatory security, quite a few MAC schemes have been introduced. The most practical solutions have been implemented and are available in commodity operating systems, such as SELinux [11],

LIDS [12], Trustees [13] and AppArmor [14]. Of these, SELinux offers the most secure solution, and has been merged into the Linux kernel for a few years. Unfortunately, despite the fact that it is available on all Linux systems, SELinux is very hard to manage: it usually requires expert skill to develop SELinux policies. Consequently, it is often recommended that novice users disable SELinux, which eventually subjects their system to vulnerabilities. But a lot of people would rather have their systems run without complexity than be secure. This is a headache for SELinux developers: how to make the system secure, but still easy to use for everyone. We believe that this problem cannot be easily solved within the foreseeable future.

In contrast with SELinux, Trustees, AppArmor and LIDS are very easy to manage, even for those who are new to these schemes. With the aim to be a simple and easy-to-use MAC solution, AppArmor enforces its policies only on selected applications (contrary to SELinux, which enforces its policies in a system-wide manner). The policy of AppArmor is based on file paths and can be declared in a human-friendly way in its policy files.

Originally a project named Subdomain [15] developed by a company called Immunix, AppArmor has been acquired by Novell and published under the open-source GPL license. Its ideas have been warmly welcomed by those who dislike the complexity of SELinux. At the moment, AppArmor attracts a group of open source developers who are working to push the project into the Linux kernel as an alternative option to SELinux.

LIDS adopts a similar approach to AppArmor, but in a “reverse” way: while AppArmor puts policies on applications, in which a policy declares which files and which modes a specific application can access, LIDS declares which applications can access a specific file. Subsequently, while LIDS seems to be more focused on protecting the data in file systems, AppArmor pays more attention to keeping applications from damaging data in file systems. Another difference is that LIDS aims to provide system-wide protection, but AppArmor only tries to protect critical applications, mostly network services. That is one of the reasons AppArmor has received criticism from the security community: if an unprotected local application is broken, the attacker can leverage it to attack a protected one, and bring it down easily.

However, AppArmor is superior to LIDS in its capabilities in confining sub processes: a thread in an application can choose to change the policy (which is called a “hat” in AppArmor terminology). As a result, AppArmor can be used to enforce separate policies for one multi-threaded application such as the Apache web server: with Apache, PHP or Perl scripts can be run with different isolated policies from the overall Apache policy.

Trustees is a MAC scheme inspired by NetWare. It allows the system administrator to attach “trustees” to any directory or file. All subdirectories and files in that directory will also inherit these trustees. Trustee rights can be overridden or extended in subdirectories. Subsequently, the access control enforced by Trustees is also based on the file path of the object.

Trustees is designed to lessen the requirement of maintaining too many ACLs in a system with a lot of files and directories. It is also hard and time-consuming to check whether all ACLs are correctly set. Deploying a system with Trustees, the administrator only needs to specify the full system rights in a small configuration file, for specific top directories.

All of the above projects are possible thanks to hooks provided by a security subsystem in Linux named the Linux Security Module (or LSM in short). LSM provides a lightweight and general-purpose access control framework for the mainstream Linux kernel. LSM enables many different access control models to be implemented as loadable kernel modules [16, 17]. Note that LSM does not provide any policy, but rather a generic framework in the form of interception points throughout the kernel. Just before the kernel would have accessed an internal object, a hook makes a call to a function provided by an LSM module. The module can either allow the access, or deny the access and force an error code to be returned.

VMAC Architecture

Goals and Approach

VMAC is designed with the aim of making it more comfortable for the administrator to handle MAC policies in a VM environment. VMAC has the following goals:

1. **An easy-to-manage security policy:** To address the difficulty of managing MAC policies in a VM environment, we propose to centralize the security policies of virtual hosts in a central machine: instead of having separate policy in each separate host, we put all the MAC policies of all the virtual hosts in an administrative VM called *Vi0*. The policies can be managed from inside *Vi0*. The reference-monitor in the kernels of these virtual hosts are modified,¹ so that MAC policies are downloaded from *Vi0*. In *Vi0*, the administrator can locally manage and update the policies of any VM at any time. Obviously, with this centralized scheme, the MAC policies become much easier to deploy in a VM environment, because everything can be done from inside *Vi0*.
2. **Aggregate and protect security logging data:** Traditionally, security logging has saved data inside each virtual host. Given machines with

¹Note that the modification is limited to the security subsystem only.

multiple virtual hosts, managing these scattered logs becomes complicated. Moreover, if an attacker breaks into a virtual host, he might compromise the logging in order to cover his penetration activities. To fix this issue, VMAC is designed to centralize security logging data: instead of keeping the logging data in each separate virtual host, all the access control logging data are sent out to *Vi0*. Thanks to this tactic, all logging data can be saved in one place, so that it is straightforward to aggregate and analyze intrusion evidence of all the maintained hosts in a site-wide manner. We suppose that in our scheme, *Vi0* is securely protected² such that the attacker has no chance to exploit our immune data, even if he totally takes over his virtual box.

VMAC Design

The architecture of VMAC exploits the fact that VMs are able to share memory with each other. So each VM (called *ViU* from now on) that wishes to have its security policies centralized and managed in *Vi0* would allocate an area of memory and share it with *Vi0*. This memory is used to exchange information: when receiving a request from *ViU* for the security policy, *Vi0* puts the corresponding policy into shared memory, and notifies *ViU* to take it. Contrariwise, when *Vi0* wants to adjust the policies of a specific virtual host, it informs the *ViU* to get rid of the old policies and take the new policies it has put into shared memory. To mitigate the performance impact, *ViU* can cache the policies in its internal kernel

²Thus *Vi0* might be considered to belong to the Trusted Computing Base (TCB), the core component required to enforce the system security policy.

memory, and only flush it out and update the cache when there are updated requests from *Vi0*. Because all information about the policies are exchanged between *Vi0* and *ViU* via the shared memory, the process incurs very little overhead, and our scheme works reliably. Note that the network stack is never used to exchange data between virtual hosts. This strategy avoids the traffic from being sniffed, which could happen if we used the network to forward data.

To send logging data to *Vi0*, all access control logging from the kernel of *ViU* is sent out to *Vi0*, and optionally sent to the traditional kernel log buffer as well. We employ the same shared memory area above for this job: the memory can be divided into two halves, with one half for exchanging security policies and the other for logging data.

VMAC consists of two main components: the first component in each *ViU*, called *VMACU*, and a daemon process running in *Vi0*, called *VMACd*. The overall architecture of VMAC is outlined in Figure 1.

The VMACU Component

The *VMACU* component in the kernel of each *ViU* is responsible for allocating kernel memory which it shares with *Vi0*. Instead of getting policies uploaded from user-space as in the legacy method, *VMACU* gets its policies updated from a daemon process named *VMACd*, which runs in *Vi0*. *VMACU* also uses this shared memory to send logging data to *VMACd*.

In order to send notifications between *VMACU* and *VMACd*, *VMACU* should employ an inter-host communication channel, and use this channel to inform the other host when it wants to get new data, or when asking the other host to get updated data put into shared memory.

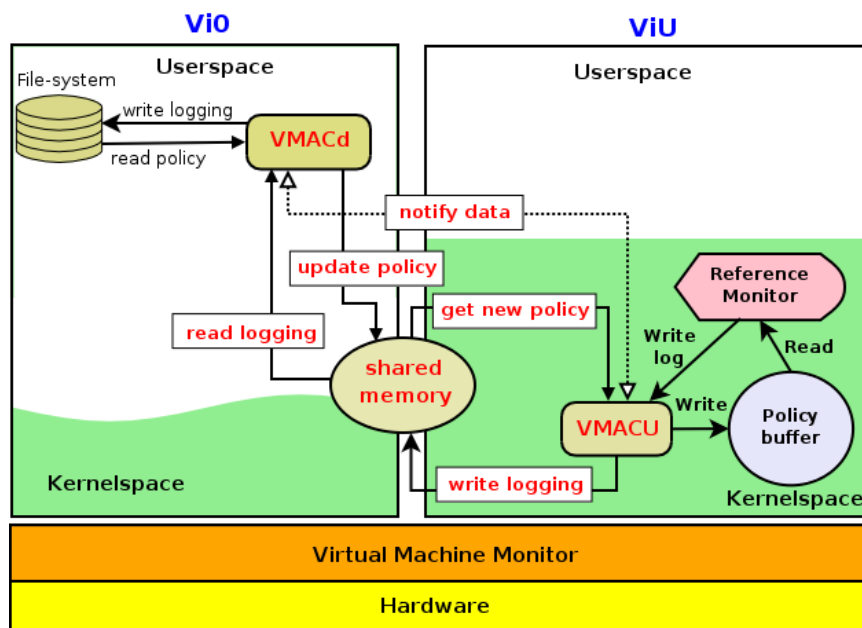


Figure 1: MAC architecture.

In our scheme, information needs to be exchanged between *VMACU* and *VMACd* in two ways: either from *Vi0* to *ViU*, in which case *VMACd* requests *ViU* to update the security policy; or from *ViU* to *Vi0*, in which case *VMACU* sends logging data out to *VMACd*. To address this problem, we employ the same trick widely used in current Xen code: a ring buffer with two halves. The first half is for sending data from *Vi0* to *ViU*, while the other half is for sending data the reverse way. Each half has two heads: one for writing data, and the other for reading data. Applying this format, the shared memory can be accessed by both sides at the same time to update and retrieve information without worrying about conflict.

When initializing, *VMACU* sends a request for security policy to *VMACd*, and gets the policy from shared memory. It then puts all the policy³ into the policy buffer, and somehow calls the reference-monitor of the MAC scheme running in the system. The reference-monitor then is notified to pick up policies from the buffer, and analyzes them. From then on, the reference-monitor caches policy in memory and never asks for the same policy again. The security enforcer does its job by retrieving policy from cache, so there is no penalty paid for getting policy from *Vi0* at runtime.

To allow administrators to update the policy, at run-time *VMACd* can also ask *VMACU* to update its policy. In this case, *VMACd* does the similar job: it copies the policy into the shared memory, and then notifies the corresponding *VMACU* to come to pick up them. This requires the MAC scheme in *ViU* to get rid of the old policy and take the new policy put in the policy buffer.

VMACU also provides a mechanism to send security logging data to *Vi0*: it exports a function for the MAC scheme to use. Whenever the MAC wishes to log data, it calls the function with the logging data as a parameter. The function then puts the data into shared memory and notifies *VMACd* to pick them up.

The VMACd Component

VMACd is a daemon which listens for policy update requests for a specific host. *VMACd* puts the policy into the shared memory of the corresponding *ViU*. Then *VMACd* informs the related *ViU* about the new policy, so that the virtual host can retrieve the updated policy.

There are two kinds of request for updating security policy that *VMACd* must serve: one is from *VMACU*, and the other is from administrators in *Vi0*.

- At initializing time, *VMACU* sends request to get the security policy. To do that, *VMACU* directly notifies *VMACd* about its demand. In this case *VMACd* passively waits for the request from *VMACU*.
- At run-time, from inside *Vi0* the administrators can modify the policy, and then asks *VMACd* to

³The policy is represented in a raw format, and VMAC does not need to understand them

send the updated policy to *VMACU*. In this case, *VMACd* actively informs *VMACU* about the updating.

In order to support both these kinds of request, we propose to let *VMACd* actively get the policy from *Vi0*. So whenever *VMACd* needs to serve the updating policy request, either to *VMACU* or administrators, it executes a *policy feeder* and gets the output of the command (this *policy feeder* tool is specific for each *ViU*). The output is regarded as the “raw policy,” which is then put into the shared memory with *VMACU* as described above.

Figure 2 describes the connection between the *policy feeder*, *vmacd* and the shared memory with *ViU*.

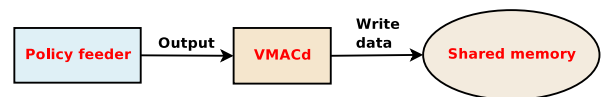


Figure 2: *vmacd* gets the raw policy from the output of the *policy feeder*, then puts these data into the shared memory with *ViU*.

Besides serving the policy update requests, *VMACd* also has the job of gathering logging data sent by *VMACU*. *VMACd* patiently waits for notification of new data from *VMACU*, then retrieves the logging data in the corresponding shared memory area and saves the collected data into separate logs for each corresponding virtual host.

VMAC on Xen

To realize the VMAC architecture described here, we have implemented on Linux with the Xen Virtual Machine. The reason we chose Xen as the virtual machine monitor is that Xen is open source, already stable and available with Linux. Note that the implementation presented here can be similarly applied to other kinds of VMs and MAC schemes, however.

Running on top of Xen, a VM is called a *Xen domain*, or domain for short. A special privileged domain called Domain0 (or Dom0 for short) always runs. Dom0 manages other domains called *User Domains* (or DomU for short), and performs tasks such as start, shutdown, reboot, save, restore and migration between physical machines. Dom0 plays the role of *Vi0*, while DomU plays the role of *ViU* in the VMAC architecture.

Because all domains run on the same machine, they can share physical resources such as memory, hardware interrupts, and peripherals. Note that all such sharing must be permitted by Xen or it is disallowed.

The following sections describe in detail the implementation of VMAC’s components in Xen.

The VMACU Component

By default, when initializing *VMACU* allocates a single page of memory (which is equivalent to 4KB in the IA32 platform) and uses this area as shared

memory with *VMACd*. Then the hardware address of the shared memory is sent to *VMACd* via *XenStore*, using the *XenBus* functions [18]. This shared memory employs the ring buffer format with two halves: one for exchanging security policies, the other for logging data, as explained earlier.

In order to send notifications between *VMACU* and *VMACd*, *VMACU* registers a Xen channel event, and informs *VMACd* about this channel via *XenStore*. *VMACU* and *VMACd* then register a virtual interrupt (called a *VIRO* in Xen terminology) and use this interrupt whenever they want to notify each other about data they put into shared memory. Information about this interrupt is also sent to *VMACd* via *XenBus* interface.

To save policies from shared memory, *VMACU* allocates a memory area and uses it as its policy buffer. After gathering policies from *VMACd*, *VMACU* calls the registered reference-monitor to retrieve the updated policies from this buffer.

Finally, to send logging data to *VMACd*, *VMACU* exports a function named *vmac_send_log()* for MAC schemes to use it. This function simply puts the logging data, which is represented by the function's parameters, to the shared memory, and then notifies *VMACd* to pick up them.

The VMACd Component

VMACd is implemented in Dom0's user-space as a daemon named *vmacd*. *vmacd* has two primary jobs: first, it must serve the policy update request on demand; second, it has to save security logging data.

Update Security Policy

One of *vmacd*'s jobs is to listen for the initializing policy request from *ViU*. Upon the request, *vmacd* executes a special *policy feeder* tool, which is specific for that DomU, and gets the output from the command. The output is considered "raw policy", and put into the shared memory with the DomU. Then *vmacd* informs the related *ViU* about the new policy, so it can retrieve the policy and update it in its cache.

vmacd also can be used as tool by administrators to update policy from Dom0 at run-time. When running with a special *--update* option on a DomU, *vmacd* does the similar things we described above: *vmacd* runs the *policy feeder* specific for that DomU,⁴ then gets the output as "raw policy" and put them into the shared memory with DomU. The final step is to notify the related *ViU* about the new policy.

To designate the *policy feeder* tool, the pathname of it must be put in a configuration file, which is specific for each DomU.

Save Logging Data

Besides the job of updating policy, *vmacd* also has to gather the logs sent from *VMACU*. *vmacd* waits

⁴this is the same *policy feeder* tool used when *vmacd* serves the request from *ViU*, explained above

for notification from *VMACU*, then retrieves the logging data in the corresponding shared memory, saving the collected data into separate logs for the corresponding domain.

To retrieve data delivered from *VMACU* and save them to logs, *vmacd* uses two separate threads: the main thread is used to get the data, and a *worker* thread is used to save the data to the file-system. The main thread silently waits for notification of new data from *VMACU*. When it detects that new data has arrived, it reads the data out from the corresponding shared memory, then copies them to a *host-buffer*⁵ allocated by *vmacd* when initializing. While the shared memory size between *vmacd* and *VMACU* should be limited (because it is the memory allocated by DomU's kernel, and kernel memory is a limited resource), this user-space memory can be much bigger.⁶ After collecting data from shared memory, the main thread wakes the *worker* up to do its job. Figure 3 outlines the diagram of the whole process.

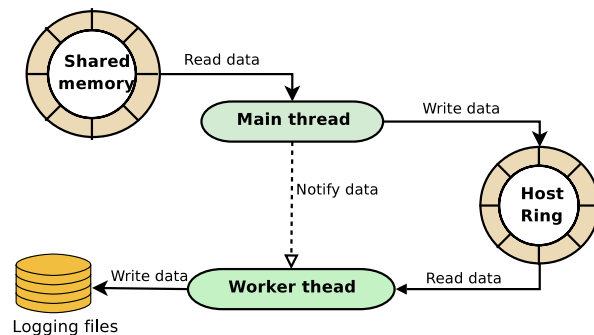


Figure 3: The *vmacd* workflow.

Regarding the *worker* thread, this thread simply waits to be woken up by the main thread (via the *pthread_cond_signal()* function), and reads the data from the *host-buffer*. The logging data is then extracted out, and saved to the logging files on file-system, separately for each domain.

Deploying MAC Schemes

The VMAC scheme presented is intended to serve as a generic framework for all sorts of MAC schemes. However, to make a specific MAC scheme work with the VMAC architecture, there is some work that must be done:

1. **Reference-monitor:** We must make the reference-monitor work with the VMAC architecture, and get the policy from *VMACU*, instead of from user-space as in the traditional method. Though *VMACU* already makes security policies available via the policy buffer, the

⁵This *host-buffer* is of the same *ring buffer* style described earlier, but we do not need two halves in this case, because data only goes one way from main thread.

⁶In fact, we set aside 32KB for this *host-buffer* area

reference-monitor must be informed to retrieve and update the policy from the policy buffer of *VMACU*. This requires a trivial modification on the reference-monitor system of MAC schemes.

2. **Logging data:** Usually MAC schemes send the logging data⁷ to the kernel buffer,⁸ and the logging data are collected by certain *syslogd* processes in user-space. We must modify the MAC schemes so they deliver these data to the shared memory between *Vi0* and *ViU* instead. In most cases, we can do this simply by replacing the original report function⁹ with the *VMACU*'s exported function *vmac_send_log()*, which puts logging data into the shared memory.
3. **Policy feeder tool:** Usually, MAC schemes keep security policy in kernel memory after loading it from user-space. Each MAC scheme employs different way to load and update the security policy. As we have seen, in the *VMAC* architecture, the *vmacd* running within *Vi0* needs to get the "raw policy" from the *policy feeder*. Fortunately, we can modify the available policy-loading and -updating tools of each MAC scheme to do this job. Typically these tools send the compiled policy directly to its kernel. We must patch them, so that they do their job via *vmacd* instead. Specifically, each original tool must be modified to play the role of the *policy feeder*, so it delivers the "raw policy" to output for *vmacd* to pick up. After that, *vmacd* transmits these special data to the corresponding *VMACU*.

Note that *vmacd* does not need to understand the semantic of the "raw policy." It just sends the data as it is, and let the MAC scheme in *ViU* do anything it wants with the received data.

To ease the burden of maintaining patches in the future, we strived to isolate and minimize our modifications to the original source code of MAC schemes. This section describes our experiences when we ported three MAC schemes – AppArmor, LIDS and Trustees – to our *VMAC* on Xen.

AppArmor

AppArmor gets its security policy via the Linux *sysfs* interfaces: loading policy is done via */sys/kernel/security/load*, updating policy via */sys/kernel/security/replace*, and removing via */sys/kernel/security/remove*. We have modified the kernel-space part of AppArmor such that all these processes are done via the policy buffer of *VMACU* instead. As a result, the original code handling the *sysfs* interfaces can be removed.

Regarding the security errors and reports made by the kernel component of AppArmor: originally

⁷Primarily, security violation reports.

⁸In the Linux kernel, reporting can be done generally with the available *printk()* function.

⁹Most are just the *printk()* function in Linux case.

they are sent to the internal kernel buffer with the *printk()* function. We modified the code so that all the messages are sent to the shared memory using the *VMACU*'s exported function *vmac_send_log()*.

At the user-space level, the tool named *subdomain_parser* originally transmitted policy data directly to kernel via these *sysfs* interfaces. We patched this tool so that it plays the role of the *policy feeder* described in the architecture of *VMACd* component above. Specifically, the modified *subdomain_parser* sends the compiled policy to standard output for *vmacd* to pick up.

All other parts of AppArmor are unchanged for *VMAC*. In total, the modifications to AppArmor are quite minimal: we replaced only 166 lines of code in the kernel-space part, and replaced around 50 lines of *subdomain_parser* with about 100 lines of new code. These modifications are pretty small when compared with the totality of the AppArmor code – version 2.0 contains around 20000 lines of code.

LIDS

LIDS employs a controversial way to load and update security policies from user-space: in contrast to the "standard method" we have commonly seen in other kernel projects, LIDS does not use an interface with user-space to load and update policy at all. Instead, LIDS exploits the system call *sys-read* to read data from user-space to the kernel. This method works quite well, and obviates LIDS from having a user-space tool to interact with kernel.¹⁰ However, this method is strongly criticized by some Linux kernel developers because it violates kernel policy by failing to work in the file-system namespace environment [19].

To adapt LIDS to *VMAC*, we modified its kernel part: we simply replaced the code that reads the security policies from the file-system¹¹ with the code to read the policies from the policy buffer. Besides, same as in AppArmor's case, we patched LIDS so that all the security messages are sent to the shared memory using the *VMACU*'s exported function *vmac_send_log()* instead of to the kernel buffer. In all, the kernel code of LIDS was changed by less than 150 lines of code.

Because LIDS user-space tools do not interact with the kernel, we had to write a small *policy feeder* that sends the compiled policy to output, which is then picked up by *vmacd*. This tool, named *lids-vmac*, is very small: only around 200 lines of C code.

Trustees

The latest stable version of Trustees is 3.0. To get security policies from user-space, Trustees creates a special virtual the kernel part of Trustees to pick up policies from the policy buffer of *vmacd*; the *trusteesfs* code can then be disabled. We also made Trustees to

¹⁰In fact, the LIDS user-space tool *lidsconf* only focuses on managing and compiling policies.

¹¹LIDS keeps all policies under the */etc/lids/* directory.

use the exported function *vmac_send_log()* to send error reports to shared memory instead of to its kernel buffer. The total code changes are around 150 lines.

As for the user-space tool *settrustees*, we modified it to be the *policy feeder*, so that it writes policies¹² to the standard output for *vmacd* to pick up, instead of to *trusteesfs*. This change affects around only 50 lines of *settrustees* code.

Discussions

We have implemented the VMAC architecture on Xen with three MAC schemes: AppArmor, LIDS and Trustees. From Xen's Dom0, the administrator is able to manage and update the policies on virtual hosts at runtime. Each DomU might be configured to run with any scheme, independent from other domains.¹³ As for the performance impact of VMAC, because our tool only modifies and adds in support for managing policy from outside, and does not change anything in the security enforcer component of the MAC schemes, VMAC causes no performance penalty on the system.

In all three ports of the mentioned MAC schemes, we have kept the interface and supported code to allow policies to be updated from the user-space of the DomU. That means the administrator could manage the security policies either from inside the DomU, or from the central Dom0. But this "dual-headed" solution can be confusing, and might cause some conflicts if the policy is updated from Dom0 at the same time. Therefore, we recommend disabling the ability to manage and update the policies from inside DomU. We provide such a compilation option to achieve this for in all three implemented schemes. This tactic has another advantage: if the DomU's kernel access is prohibited,¹⁴ if an attacker successfully breaks into the DomU, he cannot access or modify the security policies in the kernel. As a result, the whole system is more secure and tamper resistant to a root-level attack.

Conclusions and Future Work

This paper proposes an architecture named VMAC to centralize the security policies for VMs, so that managing MAC policies in this environment becomes easier and more flexible. VMAC also centralizes the security logs, so collecting and analyzing information to detect security violations becomes more straightforward. VMAC has no performance impact to the system, as the security enforcer remains unmodified inside the virtual host's kernel.

To prove the concept, we implemented VMAC on top of the Xen Virtual Machine and ported three

¹²Trustees puts all its policies in the file */etc/trustees.conf*.

¹³Unfortunately at the moment LSM does not support stacking module, so each virtual domain can adopt only one kind of scheme at a time.

¹⁴In Linux, this can be done by removing the ability to load the kernel module, together with eliminating the */proc/{mem,kmem,port}* interfaces.

popular MAC schemes – AppArmor, LIDS and Trustees – to our system. The whole system is flexible, easy to manage and can help ease the hard job put on MAC administrators.

As of this writing, Xen is being pushed into the Linux kernel, and the process is expected to be done sometime in 2006. Once Xen is merged, it will become widely used and a practical solution for everyone. We expect that our solution will then be useful for many people.

For the time being, we are making extensive investigations of other MAC techniques, such as FLASK (which SELinux is based on), RSBAC [20] and Grsecurity [21]. In the future, we plan to go on to support these other schemes if there are requests to do so.

At the moment, we have implemented *VMACU* only for Linux-based VMs. We plan to provide support for other operating systems, such as FreeBSD and NetBSD once these ports work more stably on Xen.

Acknowledgments

We are grateful for the comments we received from Trey Harris, our shepherd, as well as suggestions from the anonymous reviewers who greatly shaped our paper.

Author Biographies

Nguyen Anh Quynh is a Ph.D. student of Graduate School of Media and Governance, Keio University, Japan. His research interest includes Computer Security, Operating System, Virtualization technology and Computer Forensics.

Ruo Ando received his Ph.D. degree in Graduate School of Media and Governance, Keio University, Japan in 2006. He is a permanent researcher of National Institute of Information and Communication Technology in Japan since 2006. His research interests include secure operating system, software verification, security testbed and emulation, and automated reasoning.

Yoshiyasu Takefuji is a tenured professor on faculty of environmental information at Keio University since April, 1992 and was on tenured faculty of Electrical Engineering at Case Western Reserve University since 1988. Before joining Case, he taught at the University of South Florida for two years and the University of South Carolina for three years. He received his BS (1978), MS (1980), and Ph.D. (1983) in Electrical Engineering from Keio University.

His research interests focus on neural computing, security, electronic toys. He received the National Science Foundation Research Initiation Award in 1989, the distinct service award from IEEE Trans. on Neural Networks in 1992, the TEPCO research award in 1993, the Takayanagi research award in 1995, the Kanagawa Academy of Science and Technology research award in 1993, the best courseware award

from Asia multimedia forum in 1999, the best paper award of Information Processing Society of Japan in 1980, special research award from the US air force office of scientific research in 2003, chairman award from JICA in 2004. He has authored 25 books including neural network parallel computing in 1992, and has published more than 200 papers.

Bibliography

- [1] McAfee Avert Labs: Rootkits, Part 1 of 3: The Growing Threat, 2006, http://www.mcafee.com/us/local_content/white_papers/threat_center/wp_newappleofmalwareseye_en.pdf.
- [2] Naraine, Ryan, *Microsoft: Stealth Rootkits Are Bombarding XP SP2 Boxes*, 2005, <http://www.eweek.com/article2/0,1895,1896605,00.asp>.
- [3] Loscocco, P. A., S. D. Smalley, P. A. Muckelbauer, R. C. Taylor, S. J. Turner, J. F. Farrell, "The inevitability of failure: The flawed assumption of security in modern computing environments," *Proceedings of the 21st National Information System Security Conference*, 1998.
- [4] Bishop, M., *Computer Security: Art and Science*, Addison-Wesley Professional, 2002.
- [5] *Fedora Core Mailing List*: "Keeping SELinux on," 2006, <http://lwn.net/Articles/173812/>.
- [6] Dragovic, B., K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, R. Neugebauer, "Xen and the art of virtualization," *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.
- [7] Pratt, I., K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, A. Mallick, "Xen 3.0 and the art of virtualization," *Proceedings of the 2005 Ottawa Linux Symposium*, Ottawa, Canada, 2005.
- [8] Xen project, *Xen virtual machine monitor*, 2006, <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/>.
- [9] Clark, B., T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, J. N. Matthews, "Xen and the art of repeated research," *Proceedings of the Usenix annual technical conference, Freenix track*, pp. 135-144, 2004.
- [10] DOD 5200.28-STD: *Department of defense trusted computer system evaluation criteria*, 1985.
- [11] Smalley, S., C. Vance, W. Salamon, *Implementing SELinux as a Linux Security Module* NAI labs report, NAI Labs, 2005.
- [12] LIDS team, *Linux Intrusion Detection System*, 2005, <http://www.lids.org>.
- [13] Ruder, Andrew, *Trustees ACL*, 2006, <http://trustees.aeruder.net/>.
- [14] AppArmor team, *AppArmor project*, 2006, <http://en.opensuse.org/Apparmor>.
- [15] Cowan, C., S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, V. Gligor, "SubDomain: Parsimonious server security," *14th USENIX Systems Administration Conference (LISA 2000)*, New Orleans, LA, 2000.
- [16] Wright, C., C. Cowan, J. Morris, S. Smalley, G. Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," *Proceedings of the 11th Usenix Security Symposium*, 2002.
- [17] Smalley, S., T. Fraser, C. Vance, *Linux Security Modules: General Security Hooks for Linux*, 2003, <http://lsm.immunix.org/docs/overview/linuxsecuritymodule.html>.
- [18] Xen project, *Xen interface manual*, 2006, <http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/interface/interface.html>.
- [19] Kroah-Hartman, Greg, *Driving Me Nuts – Things You Never Should Do in the Kernel*, 2005, <http://www.linuxjournal.com/article/8110>.
- [20] RSBAC Team, *RSBAC: Extending Linux Security Beyond the Limits*, 2006, <http://www.rsbac.org/>.
- [21] Spengler, Brad, *Grsecurity*, 2006, <http://www.grsecurity.net/>.

A Platform for RFID Security and Privacy Administration

Melanie R. Rieback – Vrije Universiteit, Amsterdam

Georgi N. Gaydadjiev – Delft University of Technology

Bruno Crispo, Rutger F. H. Hofman, and Andrew S. Tanenbaum – Vrije Universiteit, Amsterdam

ABSTRACT

This paper presents the design, implementation, and evaluation of the RFID Guardian, the first-ever unified platform for RFID security and privacy administration. The RFID Guardian resembles an “RFID firewall,” that monitors and controls access to RFID tags by combining a standard-issue RFID reader with unique RFID tag emulation capabilities. Our system provides a platform for both automated and coordinated usage of RFID security mechanisms, offering fine-grained control over RFID-based auditing, key management, access control, and authentication capabilities. We have prototyped the RFID Guardian using off-the-shelf components, and our experience has shown that active mobile devices are a valuable tool for managing the security of RFID tags in a variety of applications, including protecting low-cost tags that are unable to regulate their own usage.

More philosophically, RFID technology vividly illustrates the difficulties of security administration in a world of increasingly pervasive, decentralized, low-cost, and low-power computing. Our paper thus also offers a glimpse of what system administration may be like in the future, when laymen face the responsibility to manage systems of tiny computers that they are barely aware of.

Introduction

Radio Frequency Identification (RFID) tags are remotely-powered computer chips that augment everyday objects with computing capabilities. Corporate executives tout RFID technology as a technological means to achieve cost savings, efficiency gains, and unprecedented visibility into the supply chain. Scientific researchers consider RFID technology as nothing short than an embodiment of the paradigm shift towards low-cost ubiquitous computing. In both cases, RFID tags will blur the boundaries between the online and physical worlds, allowing individuals to manage hundreds of wirelessly interconnected real-world objects, like dendrites in a global digital nervous system.



Figure 1: Philips I.Code RFID Tags.

RFID tags may be the size of a grain of rice (or smaller), and have built-in logic (microcontroller or state machine), a coupling element (analog front end with antenna), and memory (pre-masked or EEPROM). Passive tags are powered entirely by their reading devices, while active tags contain auxiliary batteries on board. Passive LF tags (125-135 kHz) can be read up to 30 cm away, HF tags (13.56 MHz) up to 1 m away, UHF tags (2.45 GHz) up to 7 m away, and active tags up to 100 m away or more.

RFID Applications and Threats

RFID automation will bring an unfathomable barrage of new applications, forever banishing wires, grocery store cashiers, credit cards, and pocket change from our lives. RFID proponents extol its professional uses for real-time asset management and supply chain management. RFID-based access passes help to police residential, commercial, and national borders; drivers have embraced RFID-based retail systems like EZ-Pass, Fast-Pass, IPass, PayPass, and SpeedPass. RFID-based “feel good” personal applications are also proliferating, from “smart” dishwashers, to interactive children’s toys, to domestic assistance facilities for the elderly. RFID tags identify lost housepets, and even keep tabs on people; the data carriers have assisted with surgeries, prevented the abduction of infants, and tracked teenagers on their way to school. Subdermal Verichips are hip accessories for patrons of several European nightclubs, and have been less glamorously deployed for identifying deceased victims of hurricane Katrina [1].

RFID technology thus races on at a pace that surpasses our ability to control it. The same ease-of-use and pervasiveness that makes RFID technology so revolutionary offers less-than-ethical characters unprecedented opportunities for theft, covert tracking, and behavioral profiling. Without the appropriate controls, attackers can perform unauthorized tag reading and clandestine location tracking of people or objects (by correlating RFID tag “sightings”). Snooping is possible by eavesdropping on tag/reader communications. Criminals can also manipulate RFID-based systems (i.e., retail checkout systems) by either cloning RFID tags, modifying existing tag data, or by preventing RFID tags from being read in the first place.

Security and privacy researchers have proposed a wide array of countermeasures against these threats. The simplest solution is deactivating RFID tags; permanently (via “frying” [17], “clipping” [13], or “killing” [4]), or temporarily (using Faraday cages or sleep/wake modes [20]). Cryptographers have created new low-power algorithms for RFID tags, including stream ciphers [6], block ciphers [5], public-key cryptographic primitives [9], and lightweight protocols for authentication [21]. Additionally, researchers have developed access control mechanisms that are located either on tag (hash locks [22] / pseudonyms [10]) or off (Blocker Tag [11], RFID Enhancer Proxy [12]).

Despite this plethora of countermeasures, neither the threats nor the fears facing RFID have dissipated. The countermeasures have become somewhat of a band-aid that can be slapped onto RFID technology later. Some companies view these results as a desirable way to quiet down the privacy activists. Other companies in RFID standardization committees are actively fighting *against* adding security into RFID protocol design, because it will make their current commercial offerings obsolete. People need a solution that they can physically own and use, not one that relies upon the RFID companies to decide when privacy will become important.

Another missing element is a means to coordinate the myriad of incompatible countermeasures as they trickle onto the market in a piecemeal fashion. Per-tag security policies combined with a lack of automation will form a management nightmare for people, who cannot be expected to know when or how to apply the appropriate countermeasures. There is no unified framework; no systematic means to leverage individual RFID countermeasures to achieve the most important goal of all – the protection of real people.

Finally, RFID technology, much like the rest of ubiquitous computing, foists system and security administration upon end users, who are likely to be neither computer literate, nor interested. The RFID Guardian thus provides a speculative look at what system administration may look like in the decentralized, low-cost, and pervasive future of computing.

RFID Guardian Design Goals

Over the past months, we have designed and prototyped the RFID Guardian, a system that allows people to administer the security of their RFID tags. The design of the RFID Guardian was driven by the following goals, which follow from the nature of RFID applications and deployment considerations:

- **Centralized use and management**

Most existing RFID countermeasures distribute their security policies across RFID tags, which make them very hard to configure, manage, and use. To address this concern, we designed a single platform to leverage RFID countermeasures in a coordinated fashion. Personalized security policies are centrally enforced by utilizing novel RFID security features (auditing, automatic key management, tag-reader mediation, off-tag authentication) together with existing ones (kill commands, sleep/wake modes, on-tag cryptography).

- **Context-awareness**

Different countermeasures have strengths and weaknesses in different application scenarios. Low-cost Electronic Product Code (EPC) tags require different access control mechanisms than expensive crypto-enabled contactless smart cards. Our system maintains both RFID-related context (i.e., RFID tags present, properties and security features, and their ownership status), as well as personal context (i.e., the user is in a non-hostile environment). Context is then used in conjunction with an Access Control List (ACL) to decide how to best protect the RFID tags in question.

- **Ease-of-use**

People do not want to fuss with an RFID privacy device, so our system must be both physically and operationally unobtrusive. We envision that our system will be eventually integrated into a PDA or mobile phone, so users will not be burdened with carrying an extra physical device. Accordingly, the RFID Guardian uses an XScale processor and simple RFID HW (barely more complex than RFID HW already found in Nokia mobile phones). Also, system operation was designed to be non-interactive for default situations, and offers a user interface for the special cases that require on-site configuration.

- **Real-world usability**

It is essential that the RFID Guardian work with actual deployed RFID systems. We chose a single standard as a proof-of-concept, to prove the technical feasibility of our ideas. Our RFID Guardian implementation supports 13.56 MHz (HF) RFID, and is compatible with the ISO-15693 [2] standard. This frequency and standard is used in a wide array of RFID applications, due to the availability of relatively inexpensive commodity HW. The ideas in this paper can also be extended to

other standards or frequencies, given some extra engineering effort.

The remainder of this paper is organized as follows. The next section describes the RFID Guardian's high-level functionality; we then provide implementation details for our RFID Guardian prototype. A real-life case study, illustrating the operation of Selective RFID Jamming follows. Subsequently, performance results are reported followed by a discussion of potential attacks. We then review some related work and conclude.

System Functionality

The RFID Guardian (first introduced in [19]) is a portable battery-powered device that mediates interactions between RFID readers and RFID tags. The RFID Guardian leverages an on-board RFID reader combined with novel tag emulation capabilities to audit and control RFID activity, thus enforcing conformance to a centralized security policy.

The vast majority of RFID readers will not explicitly interact with the RFID Guardian. Eavesdropping and clever tag emulation tactics are necessary to glean information from these readers. However, a small group of RFID readers will have special back-end SW installed, that provides them with an "awareness" of the Guardian.¹ These RFID readers tend to be in familiar locations (i.e., at home, at the office), and they are intentionally granted more generous access permissions. These RFID readers may explicitly cooperate with the Guardian, sending data containing authentication messages, context updates, or secret keys.

The rest of this section describes the design of the RFID Guardian, focusing on four fundamental issues: (i) auditing, (ii) key management, (iii) access control, and (iv) authentication.

Auditing

The RFID Guardian monitors RFID scans and tags in its vicinity, serving as a barometer of (unauthorized) RFID activity. RFID auditing is a prerequisite for the enforcement of RFID security policies, plus it furnishes individuals with both the awareness and proof needed to take legal recourse against perpetrators of RFID abuse.

Scan Logging

Scan logging audits RFID scans in the vicinity, which are either displayed (using an LCD or screen) or are logged for later retrieval. Tag emulation decodes the RFID reader queries prior to logging the 64-bit UID (tag ID), an 8-bit command code, and annotations (like a 32-bit timestamp). Query data is logged by default, unless the flash memory is almost full.

Audited RFID scans should be filtered to avoid overwhelming the user with uninteresting information.

¹Even these "Guardian aware" readers still use standard RFID hardware and air interfaces.

For example, the RFID Guardian might be configured to only log scans targeting tags "owned" by that individual (see next section). Repeatedly polled queries (like inventory queries, which ask tags in range to identify themselves) will also generate a lot of noise, so it is best to have the SW aggregate these queries (e.g., 1000x inventory query from time t1-t2).

Tag Logging

The RFID Guardian tracks RFID tag ownership and alerts individuals of newly appearing (possibly clandestine) tags. Ownership of RFID tags can be transferred explicitly via the user interface or an authenticated RFID channel (i.e., while purchasing tagged items at an RFID-enabled checkout). Ownership of RFID tags can also be transferred implicitly (i.e., when handing an RFID-tagged book to a friend.) The RFID Guardian detects implicit tag acquisition by conducting periodic RFID scans, and then correlating the tags that remain constant across time.

The frequency of RFID tag discovery is adjustable. Given that not all implicit tag acquisitions are desirable, the frequency of scanning/correlation/reporting presents a tradeoff between privacy, accuracy, and battery life. Our opinion is that infrequent correlation in a controlled environment is probably the most useful and least error prone option (i.e., comparing RFID tags present at home at the beginning and end of the day).

Key Management

Modern RFID tags have a variety of security functionality, ranging from tag deactivation commands, to password-protected memory, to industrial-grade cryptography. These security features often require the use of associated key values, which present logistical issues because the keys must be acquired, stored, and available for use at the appropriate times.

The RFID Guardian is well suited to manage RFID tag keys due to its 2-way RFID communications abilities. Tag key transfer could occur by eavesdropping on the RFID channel when a reader (for example, an RFID tag "deactivation station") issues a query containing the desired key information. Additionally, "Guardian aware" RFID readers can transfer key information explicitly over a secure channel, or key values can be manually entered via the user interface. The RFID Guardian is also an appropriate medium for periodically regenerating tag keys, re-encrypting tag data [8], and refreshing tag pseudonym lists [10].

Access Control

RFID technologists and privacy activists propose deactivating RFID tags after sale as a means of protecting consumer privacy (and corporate liability). However, if you consider that RFID tags represent the future of computing technology, this proposal becomes as absurd as permanently deactivating desktop PCs to reduce the incidence of computer viruses and phishing.

Perhaps RFID tags are in fact too much like modern computers – their default behavior is to indiscriminately transfer data to anyone with compatible equipment. The hope is that modern security technologies like firewalls and proxies can be adapted, to protect hapless RFID tags from themselves via central monitoring and managing of the communications medium.

Coordination of Security Primitives

The RFID Guardian maintains a centralized security policy that dictates which RFID readers have access to which RFID tags in which situations. This security policy is implemented as an Access Control List (ACL). The ACL resembles one used by a standard packet filter, that allows or denies RFID traffic based upon the querying reader (if known), the targeted tag(s), the attempted command, and the context (if any).

Permitted data types in the ACL are values (i.e., 123), text strings (i.e., ‘at home,’ ‘in a paranoid mood’), groupings (i.e., assigned groups of tags/readers/context/commands), and wildcards (123*, *). The user configures the ACL, and constructs the groups via the user interface.

Context-awareness

Different situations call for different countermeasures. For example, RFID tagged credit cards require less stringent security at home than at the shopping mall. The RFID Guardian therefore offers context awareness facilities that perceive an individual’s situation and then regulate tag access accordingly.

Well defined context like dates and times are easy to infer, but are marginally useful for describing a person’s situation, moods, or desires. Alternately, more abstract context information can be represented via “context updates,” which are arbitrary textual strings that represent some facet of the user’s situation. Context updates could report anything. For example, an RFID reader at the front door of a person’s home might inform the RFID Guardian that it is now leaving a protected area. Context updates are provided either by user (via the user interface), or by authenticated “Guardian aware” RFID readers.

Tag-reader Mediation

The RFID Guardian acts as a mediator between RFID readers and RFID tags. Just like a packet filter, the Guardian uses Selective RFID Jamming [18] to enforce access control by controlling the communications mediation. The RFID Guardian can therefore control access for low-cost RFID tags that otherwise might not have any access control primitives available to them.

The RFID Guardian’s selective jamming scheme is currently optimized for ISO-15693 tags, which use the Slotted Aloha anticollision scheme (as opposed to EPCglobal’s ‘tree-walking’). Selective RFID Jamming uses tag emulation to decode the incoming RFID reader query, determines if the query is permitted (according to the ACL), and then sends a short jamming signal that

precisely blocks the timeslot in which the “protected” RFID tag will give its response.

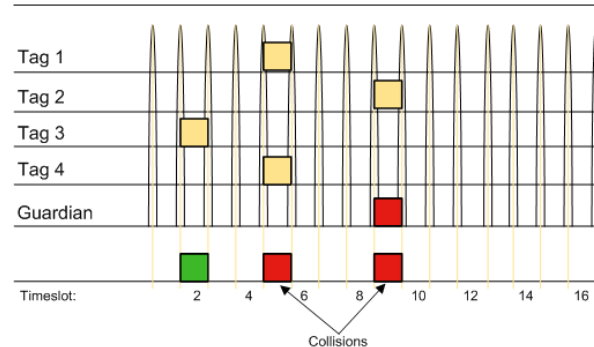


Figure 2: Selectively Jamming Tag #2.

There are 16 timeslots after an inventory query, so during the first round of anticollision, the jamming has a 1 in 16 chance of accidentally interfering any other RFID tag present. During each subsequent round of anticollision, the reader issues another inventory query with a slightly modified mask value, that targets a slightly narrower range of RFID tags than before. Given enough rounds of anticollision, the mask value will exclude the RFID tag(s) that are being “protected,” allowing other tags in the vicinity to get their responses heard by the RFID reader. This means that in practice, our system has a negligible chance of blocking the incorrect RFID tag responses. This makes the RFID Guardian’s manner of selectively jamming inventory queries far less-obtrusive than the Blocker Tag’s concept of “privacy zones” [11], which block entire ranges of tag identifiers (regardless of who owns the tag.)

Authentication

Some high-cost RFID tags can directly authenticate RFID readers, but the majority of RFID tags cannot due to application constraints (i.e., cost or power). The RFID Guardian thus authenticates “Guardian aware” RFID readers on behalf of low-cost RFID tags, adapting the subsequent access control decisions to reflect the permissions of the newly-identified reader. Prior to authentication, the RFID Guardian must also exchange authentication keys with RFID Readers, either ahead of time or using on-the-fly means (ex. user interface, PKI).

After the successful authentication of a reader, the RFID Guardian faces a practical problem: for non-cryptographic RFID tags there is no easy way to determine which RFID queries originate from which RFID reader. The best solution would be for RFID standardization committees to add space for authentication information to the RFID air interface. However, until that happens, we are using our own imperfect solution: in the last step of authentication an RFID reader announces which queries it’s going to perform, and these queries are noted as part of an “authenticated session” when they occur.

Implementation

The RFID Guardian prototype, shown in Figure 3, is meant to help people solve their RFID privacy problems in a practical way. Therefore, we have tested our system against commonly used RFID equipment – the Philips MIFARE/I.Code Pagoda RFID Reader, with Philips I.Code SLI (ISO-15693) RFID tags. This section will introduce the hardware and software architecture that our prototype uses to monitor and protect the RFID infrastructure.

Hardware

The RFID Guardian hardware architecture is presented in Figure 4.

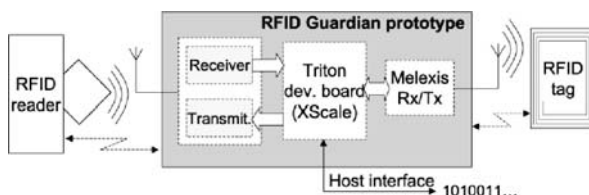


Figure 4: RFID Guardian HW Architecture.

Our first salient design decision was to make the RFID Guardian a full-fledged portable computer. We chose a “beast” of a microcontroller – the Intel XScale PXA270 processor, with 64 megabytes of SDRAM and 16 megabytes of Flash memory. We rationalized the use of the XScale by the strict ISO-15693 timing constraints combined with the computational load of authenticating RFID readers. (A later section analyzes the extent to which the PXA270 is overkill.) Another benefit of the XScale processor family is its wide deployment in handheld devices, which eases eventual integration of the RFID Guardian into PDAs and mobile phones.

Our prototype has a minimalist User Interface (UI) at the moment – a serial RS-232 interface to the PC host, which contains an attached keyboard and screen. While this is sufficient for our proof-of-

concept, we plan to add a more portable UI to the next version of the RFID Guardian HW.

RF Design Overview

The analog part of our prototype consists of an “RFID reader” front end that uses an RFID reader-on-a-chip, and an “RFID tag” front end which required building our own custom tag emulation HW.

Our *reader transmitter/receiver* was implemented using an ISO-15693 compliant RFID reader IC from Melexis (MLX90121) [16] together with a power stage, based on the application note AN90121_1 [15], that increases the operating range to 30 cm.

Our *tag receiver* is based on an SA605 IC from Philips. The IC is intended for a single chip FM radio, but we used it to implement a high sensitivity AM receiver. Because our receiver is battery powered (as opposed to passively-powered RFID tags), it receives RFID reader signals up to a half meter away.

Our *tag transmitter* implements “active” tag spoofing using an RF power stage and a dedicated digital part that generates and mixes the required sideband frequencies, 13.56 MHz +/- 423 kHz. By actively generating the sideband frequencies, we can transmit fake tag responses up to a half meter.

We also use our tag transmitter as the basic HW primitive to generate the RFID Guardian’s randomized jamming signal. (This is described further in the SW section.)

Tag Spoofing Demystified

RFID readers produce an electromagnetic field that powers up RFID tags, and provides them with a reference signal (e.g., 13.56 MHz) that they can use for internal timing purposes. Once an RFID tag decodes a query from an RFID reader (using its internal circuitry), it encodes its response by turning on and off a resistor in synchronization with the reader’s clock signal. This so-called “load modulation” of the carrier signal results in two sidebands, which are tiny peaks of radio energy, just

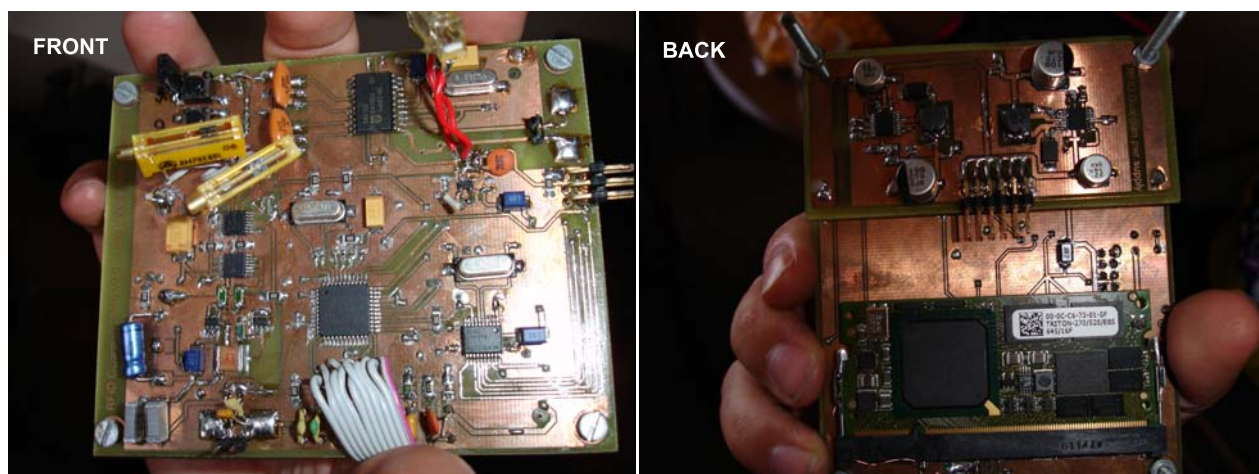


Figure 3: RFID Guardian Prototype.

higher and lower than the carrier frequency. Tag response information is transmitted solely in these sidebands,² rather than in the carrier signal.

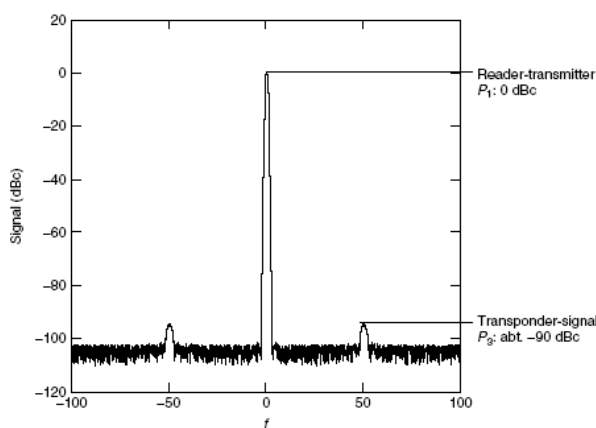


Figure 5: Normal RFID Tag Signal.

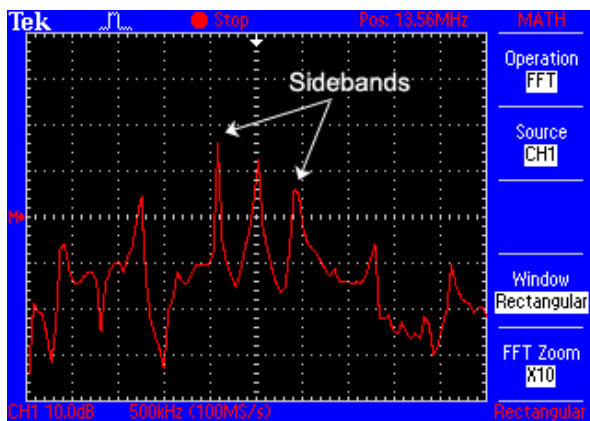


Figure 6: Spoofed RFID Tag Signal.

Figure 5 (from the RFID Handbook [6]) illustrates how these sidebands look, in relation to the reader-generated carrier frequency. The comparatively tiny sidebands have approximately 90 decibels less power than the reader-generated carrier signal, and this is the reason why RFID tag responses often have such a limited transmission range.

The secret to creating fake tag responses is to generate the two sideband frequencies, and use them to send back properly-encoded responses, that are synchronized with the RFID reader's clock signal. The simplest way to generate these sidebands is to imitate an RFID tag, by turning on and off a load resistor with the correct timing. The disadvantage of this approach is that passive modulation of the reader signal will saddle our fake tag response with identical range limitations as real RFID tags (~10 cm for our test setup).

²Sidebands are not just an RFID-specific phenomenon – they are also commonly used to transmit information in radio and television broadcasts, long-distance voice communications, and amateur radio.

A superior alternative is to use battery power to generate the two sideband frequencies. These super-powerful sidebands are detectable at far greater distances, thus increasing the transmission range of our fake tag response.

The RFID Guardian prototype utilizes the “active” tag spoofing approach. Figure 6 shows the signal generated by our tag transmitter. The spoofed “sidebands” are transmitted at a power-level roughly equal to the reader's carrier signal. This has increased the range of our fake tag responses – from 10 cm to a half meter away!

Software

The RFID Guardian is like a watchdog; it sits with a cocked-ear, waiting for danger to appear. It monitors real-world activity, from unexpected RFID scans to clandestinely located tags, and reacts in real-time lest these dangers remain undetected and undeterred.

The RFID Guardian's SW architecture reflects this event-driven reality. Besides its real-time core, the Guardian's 12694 lines of code provide device drivers (for our RFID HW), a protocol stack (ISO-15693), data storage libraries, high-level system tasks, and application libraries. The result is 254728 bytes of cross-compiled functionality dedicated to RFID security and privacy protection.

Operating System

The RFID Guardian presents a holistic system to users, but lurking below the surface are time-critical SW routines that require central coordination. The e-Cos Real-Time Operating System (RTOS) takes the place of taskmaster; it ensures fast and reliable execution, while simplifying developers' lives by handling threads, basic common interrupt handling, and some device drivers (i.e., RS-232 driver). e-Cos was selected primarily for its availability for the PXA270 microcontroller, but it also proved an excellent choice because it is open-source, free of licensing costs, and has an active developer community.

Libraries

A major portion of the RFID Guardian SW handles intermediate processing steps; e.g., tag spoofing requires ISO-compliant frame modulation and encoding, and scan logging requires a mechanism for caching data in the Flash memory. This section will describe the low- and medium-level libraries that support the main RFID Guardian functionality.

Device Drivers: Device drivers are the steering software for the RFID Guardian's HW. Driver pairs control the RFID tag device (tag transmitter/receiver), RFID reader device (reader transmitter/receiver), and the jamming signal (random noise generated by the tag transmitter). Device drivers can read/write bytes and RFID markers (EOF, SOF, JAM), and they can also provide timing information. eCos also conveniently provides device drivers for the RS-232 “user

interface,” which facilitates a connection to the user’s keyboard and screen.

Protocol Stack: Once the device drivers decode bytes of raw RFID data, the RFID Guardian needs to make further sense out of it; e.g., was it an RFID tag replying to an inventory query, or an RFID reader attempting to read a data block? The ability to understand RFID communications protocols is a prerequisite for making meaningful high-level security decisions (e.g., was the reader’s read command authorized?) This is why the RFID Guardian contains an implementation of Part 2 (device drivers) and Part 3 (Communications protocol) of the ISO-15693 standard.

Data Storage Once RFID communications have been interpreted, the internal state of the RFID Guardian is updated by modifying the contents of one or more data structures. Generally, this data is stored in the volatile RAM, but “permanent” data structures are cached into Flash when the processor is idle. The Journaling Flash File System (v2) manages the RFID Guardian’s Flash memory, providing filesystem-style access, offline garbage collection, balanced erasing of blocks, and crash resistance.

The data structures themselves collectively reflect the high-level functionality of the RFID Guardian. Transient data structures include the tag presence list, partially-open authentication list, authenticated session list, context list, and timer activity list. Permanent data structures may also include the RFID scan log, access control list, reader authentication key list, tag ownership list, and tag key list.

Tasks

The RFID Guardian’s high-level system tasks are little virtual pieces of functionality that take turns controlling the behavior of the system. Each task plays a different role: the tag task acts like a virtual RFID tag, and the reader task like a commodity RFID reader. The timer task is akin to a little alarm clock, that periodically goes off and spurs other system components into action. The user input task primarily relays input from the real-life user input devices to the appropriate SW handler.

Each of these tasks uses a comparable software stack. A main loop at the top level waits for activity on any device, and an interrupt prompts the device driver to decode and store the frame(s). The task then invokes the appropriate high-level application routines.

Timer Task The RFID Guardian needs to perform activities at specific times, either periodically (i.e., polling to populate the RFID tag presence list), or on a one-time basis (i.e., timing out a half-opened authentication attempt). The timer task is responsible for keeping track of scheduled activities, and multiplexing the XScale’s high-resolution timer interrupts with the corresponding actions that must occur at those times.

User Input Task: On rare occasions, users will want to explicitly interact with the RFID Guardian.

They may want to configure the ACL, conduct an RFID scan, provide context data, or execute some other kind of system command. The user input task collects these commands from the cornucopia of available input devices, (i.e., RS-232, keyboard/button/keypad/etc.), and reroutes them to the system components responsible for the desired high-level functionality.

Tag Task: Tag emulation is one of the highlights of the RFID Guardian, being frequently used to achieve the RFID Guardian’s high-level goals – RFID scan logging, authenticating RFID readers, and spoofing one or several RFID tags. The tag task is the entity responsible for coordinating the RFID Guardian’s “tag-like” behavior. When activated by an interrupt from the tag receiver, the task calls the device driver to demodulate and decode the incoming RFID queries. This subsequently activates the aforementioned high-level functionality, if needed.

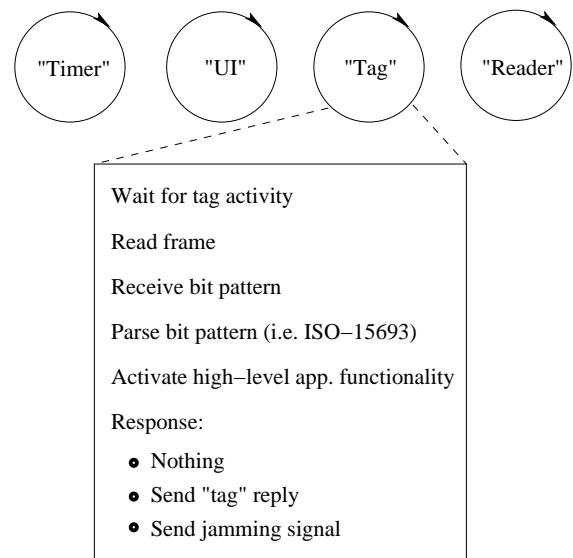


Figure 7: ‘Tag’ Task Functionality.

Reader Task: The reader task, driven by SW requests from the timer and UI, coordinates use of the Guardian’s RFID reader-on-a-chip. The task performs specified queries, (i.e., inventory, read/write data), and interprets the tag responses. This is commonly used for detecting (possibly covert) RFID tags, and activating on-tag security mechanisms, if any.

Inter-Device Functionality

Lots of high-level application functionality has been introduced in this paper, but little has been said about the RFID Guardian’s interactions with “Guardian aware” RFID infrastructure (introduced in the second section).

RFID Guardian-Reader communications use a meta-language that we call Guardian Language (GL), which is encapsulated in standard ISO-compliant ‘read/write multiple blocks’ commands. GL uses an 8-bit

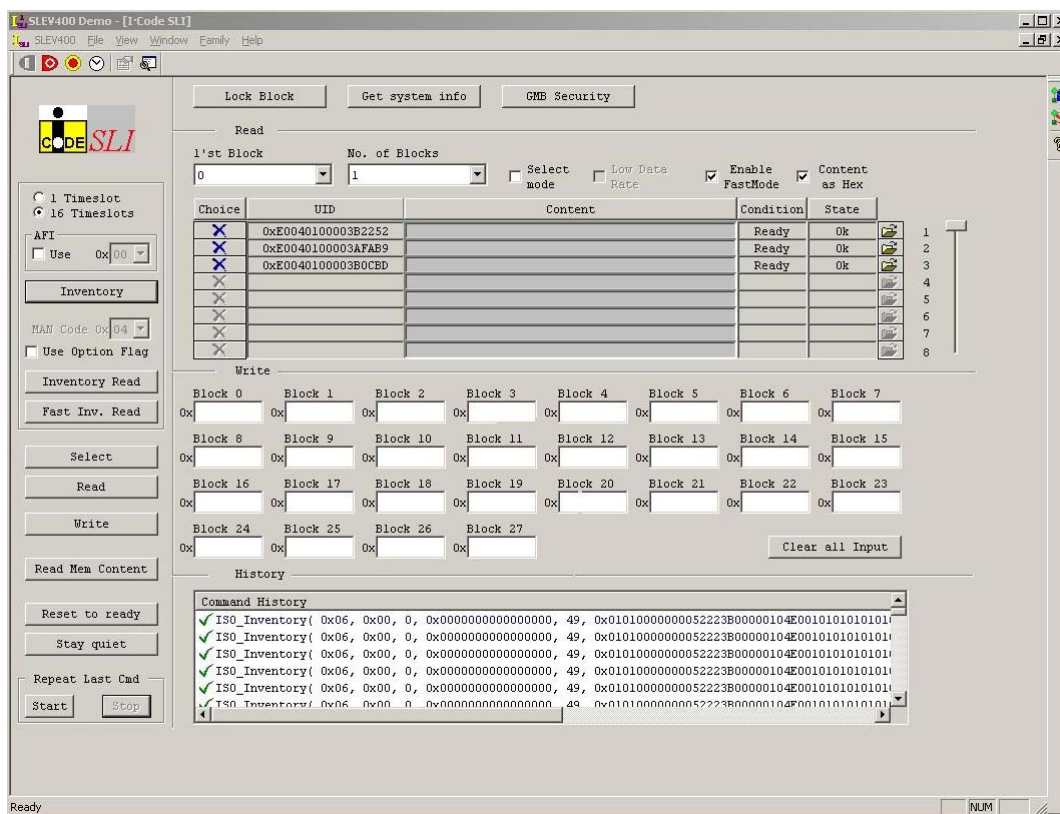


Figure 8: Screenshot During Uninterrupted Query.

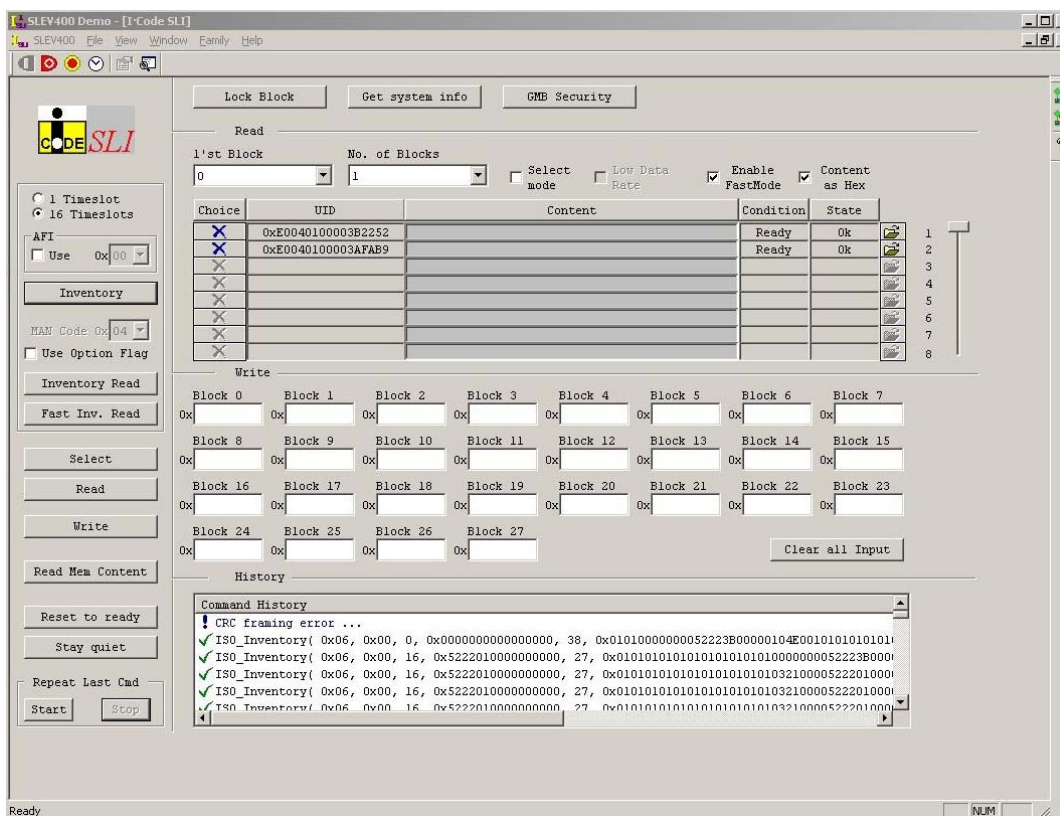


Figure 9: Screenshot During Selective RFID Jamming.

Distinctive Starting Block, an 8-bit GL Command and a varying amount of Command Data. The theoretical length limit for command data is 8 kBytes, although the practical limit is 128 bytes, which is the capacity of our I.Code SLI tags.

Here is how GL looks when encapsulated a ‘read multiple block’ response:

SOF	Flags	DSB	GLC	Command Data	CRC16	EOF
	8 bits	8 bits	8 bits	256 bits – 64 kbits	16 bits	

Here is a non-exhaustive list of GL commands: Initiate Authentication, Authentication Response, Key Update, Forward Query (proxy mode), Add Tag, Remove Tag, Add Reader, Remove Reader, and Context Update. GL also features non-standard configuration commands, that require some knowledge about the RFID Guardian internal setup.

One caveat is that, because the RFID Guardian is emulating an RFID tag, Guardian-Reader communications are constrained by master-slave interactions. In other words, RFID readers must always initiate communications with the RFID Guardian. Designers must keep this in mind when creating interaction patterns for new RFID security and privacy functionality.

Case Study: Selective RFID Jamming

This section will provide a step-by-step demonstration of how Selective RFID Jamming works.

For demonstration purposes, we have given the RFID Guardian a minimal tag ownership list that contains only one tag (UID: 0xe0040100003b0cbd). A single entry in an equally minimal ACL prescribes blocking all tags in the ownership list:

Tag	Reader	Command	Context
...
<ownership list>	*	*	*

We now generate inventory queries with our Philips MIFARE/I.Code Pagoda RFID Reader, which is driven from a Windows PC interface. Initially the RFID Guardian is switched off, and the Philips Reader detects three tags in its vicinity: the one tag that is in our ownership list, and two unknown tags (UID: 0xe0040100003b2252 and 0xe0040100003afab9). (See Figure 8 for a screenshot.)

When the RFID Guardian is enabled, the Philips Reader’s inventory queries are immediately detected. These requests are decoded, and the RFID Guardian’s internal logic determines that the query should be blocked. The Guardian then sends a short (ca. 350 μ sec) jamming signal at timeslot 13 of the inventory sequence, since that slot corresponds to the protected tag: 0xe0040100003b0cbd.

Only the two unprotected tags are recognized by the Philips reader now, and the jamming caused a

CRC error that is reported in the lower central pane of the reader’s user interface (see Figure 9).

Debug output from the RFID Guardian illustrates the processing steps, including the decision to jam at timeslot 13:

```

1 Request t_eof 76.877230 RFID_INVENTORY(
1a flags=RFID_FRAME_DATA_RATE_FLAG|
1b RFID_FRAME_INVENTORY_FLAG),
1c masklen=0x00,mask=0x0;
2 Inventory: t_eof 76.877230 s->SN 0 s->NbS 16
3 Inventory: t_eof 76.882010 s->SN 1 s->NbS 16
4 Inventory: t_eof 76.886791 s->SN 2 s->NbS 16
5 Inventory: t_eof 76.888304 s->SN 3 s->NbS 16
6 Inventory: t_eof 76.891568 s->SN 4 s->NbS 16
7 Inventory: t_eof 76.896340 s->SN 5 s->NbS 16
8 Inventory: t_eof 76.901120 s->SN 6 s->NbS 16
9 Inventory: t_eof 76.905893 s->SN 7 s->NbS 16
10 Inventory: t_eof 76.910673 s->SN 8 s->NbS 16
11 Inventory: t_eof 76.915446 s->SN 9 s->NbS 16
12 Inventory: t_eof 76.920225 s->SN 10 s->NbS 16
13 Inventory: t_eof 76.924999 s->SN 11 s->NbS 16
14 Inventory: t_eof 76.929778 s->SN 12 s->NbS 16
15 Inventory: t_eof 76.934552 s->SN 13 s->NbS 16
16 Inventory JAM t 76.934869 on s->SN 13 s->NbS 16
16a mask len 0 mask 0x0
17 Inventory: t_eof 76.939330 s->SN 14 s->NbS 16
18 Inventory: t_eof 76.944107 s->SN 15 s->NbS 16

```

Lines 1-1c report an Inventory request with a mask length 0, and flags indicating a 16-slot inventory sequence. Lines 2 through 18 report End of Frame (EOF) pulses that mark the start of a new timeslot. (s->SN indicates the current slot number.) Line 16-16a corresponds with timeslot 13, and it indicates the generation of a jamming signal.

Performance Measurements

This section will analyze the performance of the RFID Guardian, under a variety of resource constraints and attack modes.

Timing Constraints

The RFID Guardian enforces access control decisions on the behalf of RFID tags, so real-time performance is required under both normal and hostile conditions. After all, blocking a tag response *after* it has reached the attacker is not very useful.

In the upper time-line of Figure 10 we show the timing constraints for an inventory request-response sequence as specified by the ISO standard. Like every other RFID message, the request is framed by a start-of-frame marker (SOF) and an end-of-frame marker (EOF). Between these markers, an inventory request carries between 40 (mask size is 0) and 104 (mask size is 64) data bits. After receiving the request EOF, the tag must wait for 320.9 μ sec before starting its answer. This is the time the RFID Guardian has to interpret reader requests and respond to them.

The lower time-line of Figure 10 shows the measured performance of the RFID Guardian. After a complete frame is received (SOF, data, and EOF), it needs 23 μ sec to wake up the thread that monitors the

receiver and parses the request frame. Immediately before dispatching the response frame, another 5 μsec of overhead is spent in firing up the transmitter. In between these two events, the RFID Guardian has $320.9 - (23 + 5) = 292.9 \mu\text{sec}$ to consult its ACL (and supporting data structures) and decide whether or not to block the RFID tag response.

How long this decision takes depends on how the RFID Guardian's ACL is organized. To find a coarse upper bound on the ACL length that can be handled by the Guardian prototype, we chose the slowest possible implementation for the ACL: an unsorted array of UIDs that can only be traversed sequentially to locate a specific UID. An RFID request addressed to the last item in the ACL was sent to the Guardian, forcing it to traverse the entire list. With 2600 entries, the Guardian was able to respond in time.

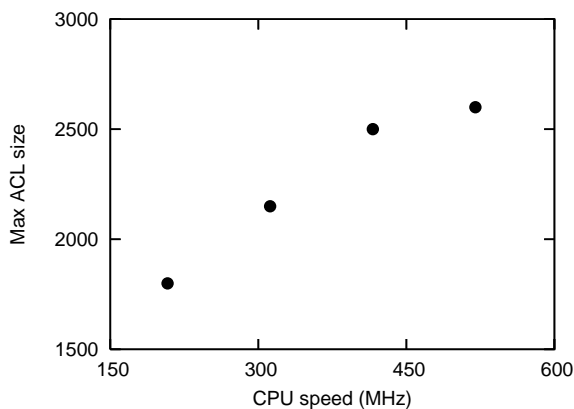


Figure 11: Maximum ACL size that can be processed at a given CPU speed.

The Guardian prototype is equipped with a powerful XScale processor at high clock speed, 520 MHz. To find out if a Guardian with less processor power would still be feasible, we varied the clock speed of the XScale. The results are shown in Figure 11. The ACL length that the Guardian could still cope with decreases with clock speed, but much less than linearly. This is attributed to two causes: memory speed goes down more slowly and in coarser steps than CPU speed; and parts of the device processing are independent of CPU speed. At 208 MHz, the Guardian prototype can process ACLs of length 1800, even with this suboptimal ACL implementation.

Of course, with a hash table instead of a linear list, vast numbers of ACLs can be searched in the available 292.9 μsec . In short, ACL length is not likely to be a problem even on a very slow XScale.

DoS Resistance

Now let us consider how attackers will try to defeat the RFID Guardian. They may use malicious readers or fake tags that try to confuse or lock up the RFID Guardian, so that the tags it protects can be read anyway. The primary defense against well-known exploits like buffer overruns must be very careful programming of the RFID Guardian software, which is helped by its limited code size.

Failing that, their next attack is likely to be a DoS (Denial-of-Service) attack to overload the RFID Guardian and prevent it from doing its job. Two RFID Guardian resources are obvious candidates for attack: its limited radio bandwidth and its limited memory. RFID communications always follow the master-slave pattern, where the tag (slave) must respond after a well-defined delay. Attacking during this delay is not feasible: it would immediately alert the RFID Guardian and it would confuse the tags as well. Attacking between reader commands does not constitute a DoS vulnerability of the communication channel: it would be the same as a regular reader action. The attacker could jam the channel, of course, but then he could not read out any tags, which is the presumed reason he wants to cripple the RFID Guardian.

The other potential vulnerability is the limited RFID Guardian flash memory. An attack on the flash memory may target any one of three data structures: the tag ownership list, the tag presence list, or the scan audit log. If an attacker with a battery-powered device simulated thousands of new tags in an attempt to fill up the ownership list or the current list, the RFID Guardian could warn the user about this abnormal activity.

Alternatively, the DoS attacker could try to fill up the audit logs. This does not cause a loss in protection of the owner's tags, but it certainly hampers the RFID Guardian's auditing capabilities. The maximum rate at which requests can be launched is determined by the bandwidth of the radio channel and the minimum frame size, both of which are specified by the standard. The data rate is 26.48 kbps. The minimum frame is (SOF, 32 data bits, EOF) which takes 1.322 ms followed by a mandatory silence of 320.9 μs , which works out to a maximum of 613 requests/sec.

An audit log entry contains the index of the tag being targeted, an index of the context, the command and a timestamp, which results in $2+2+1+4 = 9$ bytes. With 613 requests/sec, the attacker can fill up 5517 bytes of flash memory per second. The RFID Guardian prototype has 16 MB of flash, of which 14

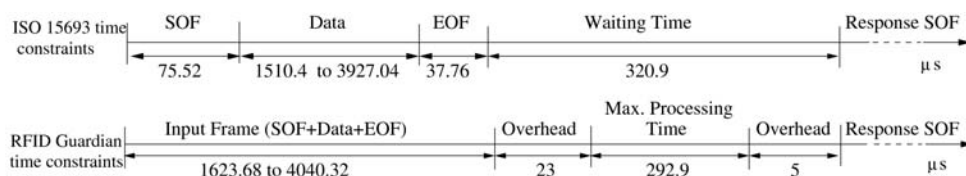


Figure 10: Timing constraints.

MB is available for logging. Thus a maximum-speed attack would need 42 consecutive minutes of blasting away at full speed to fill the memory. Needless to say, the RFID Guardian should be sounding an alarm long before the memory begins to fill up, thus fulfilling its job of warning the user of an attack. Besides, flash memory is very cheap: another 16 MB might would add less than 2 dollars to the production cost.

To summarize, the RFID Guardian seems immune to the DoS attacks that we can identify, either because they would also disturb regular RFID interaction, or because the RFID Guardian has enough resources to defend itself long enough to alarm its owner after the threat has continued for some while.

Discussion

In contrast to the aforementioned Denial of Service attacks, there are a number of attacks that are successful against the RFID Guardian.

The RFID Guardian faces the ‘hidden station’ problem, which is a geometric problem that depends entirely upon radio ranges. However, we assume that an attacker wouldn’t be able to maintain this for long, so we only deal with the “single reader” problem in this paper.

RFID readers could potentially trace the collision space, using collisions to resolve the IDs of RFID Guardian-protected protected RFID tags. We can improve this situation by adding some extra collisions, which will cause the algorithm to traverse a greater part of the ID space, making it look like more than one protected tag is present.

Another weakness of the RFID Guardian is its inability to jam reader queries. Selective RFID jamming only jams tag responses – not queries. However, queries can modify an RFID tag in unauthorized ways, like performing unauthorized data writes, or tag “killing.” Other mechanisms can protect RFID tags from this, like temporary tag deactivation PETs (i.e., sleep/wake modes). However, this remains problematic for low cost RFID tags that might not support these other modes.

Finally, attackers can evade RFID Guardian protection by tracking people using tags with pseudonyms.

If the RFID Guardian has the pseudonym list (or PRNG seed), it can correlate the IDs, remaining aware that it is dealing with only one tag. If the RFID Guardian doesn’t have the list (or seed), it will think that it is dealing with multiple tags that are only observed once. The RFID Guardian also has trouble dealing with tags working with unknown standards/frequencies.

Related Work

Given how great the threat of RFID technology is to privacy, it is not surprising that other researchers are also thinking about privacy defenders. Probably the closest work to ours is the RFID Enhancer Proxy [12], which shares some similarities with the RFID Guardian. The REP, too, is an active mobile device that performs RFID tag security management, using a two-way communications channel between the REP and RFID Readers. However, the REP has some key differences from the RFID Guardian. The most important differences are as follows. First, the REP explicitly “acquires” and “releases” RFID tag activity, which the Guardian does not require. Second, the REP’s two-way communications channel is “out-of-band,” which requires extra infrastructure. Third, the “tag relabeling” mechanism requires RFID tags to generate random numbers (or have a sleep mode), which many of them cannot do (or do not have). Fourth, the REP is purely theoretical; in contrast the RFID Guardian has been implemented and tested.

RFID tag auditing (and cloning) are supported by several devices. FoeBuD’s Data Privatizer [7] will detect RFID scans, find and read RFID tags, and copy data read to new tags. The Mark II ProxCard Cloner, by Jonathan Westhues [23] is a more general-purpose proximity-card cloner, that supports the emulation of several RFID frequencies and standards (the HW is elegant, but the SW is pending). Neither of these perform all the auditing, key management, access control, and authentication functions that the RFID Guardian does.

A less sophisticated approach to privacy protection is to block scans irrespectively of their originating reader. The Blocker Tag (Juels) [11] originated the concept of ‘RFID blocking’ as a form of off-tag access

Tool Name	Tag emulation (SW)	Tag emulation (HW)	Scan auditing	Access control	Authentication	Implementation
NFC	✓					✓
Data Privatizer	✓		✓			✓
Blocker Tag	✓			✓		✓
Field Probe	✓	✓	✓			✓
ProxCard Cloner	✓	✓	✓			✓
RFID Enhancer Proxy	✓		✓	✓	✓	
RFID Guardian	✓	✓	✓	✓	✓	✓

Table 1: RFID Tag Emulators for Security/Privacy.

control. It is designed to abuse the tree-walk anticollision protocol, and RFID readers are forced to traverse the entire id namespace when trying to locate RFID tags. This approach does not analyze incoming scans, look up information in an access control list, and depending on what it finds, take action as the RFID Guardian does. Also, it has not been implemented. (A purely SW-based “soft” blocker tag has been implemented, but it expects RFID readers to self-regulate their behavior.)

An active device that can detect RFID scans is the M.I.T. RFID Field Probe [14]. It is a portable device, created by Rich Redemske at MIT Auto-ID Center, that integrates an RFID tag emulator and sensor probe. The HW consists of a semi-passive tag, a power level detector, and a helper battery. The RFID field probe gives audio and visual representations of the field signal strength and signal quality. However, its function is not to protect its owner’s privacy, but as a tool to help vendors determine where on their pallets to attach the RFID tag to maximize signal strength for supply-chain management applications. Consequently, it does not have anything like our software, which is the heart of the RFID Guardian’s privacy defense.

Several other RFID-based technologies support the concept of two-way RFID communications. Near Field Communications [3] is a peer-to-peer RFID-related communications technology. NFC devices can query RFID tags, and can also communicate with other NFC-enabled devices. However, NFC devices cannot talk with non-NFC enabled RFID readers and do not do privacy protection.

Finally, the RFID countermeasures described in Section 1.1 are all complimentary to the RFID Guardian, in the sense that the RFID Guardian could leverage them as part of its framework, for helping to provide personalized access control. However, none of them are discrete devices that protect privacy.

Conclusion

If we are ever immersed in a sea of RFID chips, the RFID Guardian may provide a life raft. This battery-powered device, which could easily be integrated into a cell phone or PDA, can monitor scans and tags in its vicinity, warning the owner of active and passive snooping. It can also do key management, handle access control, and authenticate nearby RFID readers automatically, taking its context and location into account, for example, acting differently at home and on the street. Furthermore, it can manage access to tags with sensitive content using Selective Jamming. No other device in existence or proposed has all of these capabilities. The RFID Guardian thus represents a major step that will allow people to recapture some of their privacy that RFID technology is threatening to take away.

However, what we have described here is only one step. We intend to further develop and improve the

RFID Guardian by giving the prototype more capabilities. These capabilities include support for more frequencies and standards, improving the communication range, and simplifying the HW design. We also intend to further develop the security protocols that are needed for the authentication and key management facilities, thinking particularly about interaction requirements with the surrounding RFID infrastructure.

On a more abstract level, the RFID Guardian addresses some of the difficulties of security administration in a world of pervasive, decentralized, low-cost, and low-power computers. Therefore, our paper not only offers a solution to a practical modern-day problem, but also provides a sense of how system administration may look in the future world of ubiquitous computing.

Acknowledgments

The authors would like to thank Serge Keijser, Tim Velzeboer, Dimitris Stafylarakis, and Chen Zhang for their technical contributions. We also thank Anton Tombeur, Eduard Stikvoort, and Koen Langendoen for their friendly advice and help.

This work was supported by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), as project #600.065.120.03N17.

More information is available at the RFID Guardian project homepage at: <http://www.rfidguardian.org/>.

Author Biographies

Melanie R. Rieback is a Ph.D. student at the Vrije Universiteit Amsterdam in the Computer Systems Group. Her research interests include computer security, ubiquitous computing, and Radio Frequency Identification. Melanie has an MSc. in computer science from the Technical University of Delft, and in a past life, she worked as a bioinformaticist on the the Human Genome Project. Contact her at Dept. of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands; melanie@cs.vu.nl; www.cs.vu.nl/~melanie.

Georgi N. Gaydadjiev was born in Plovdiv, Bulgaria, in 1964. He is currently an assistant professor at the Computer Engineering Laboratory, Delft University of Technology, the Netherlands. His research and development experience includes 15 years in hardware and software design at System Engineering Ltd. in Pravetz Bulgaria and Pijnenburg Microelectronics and Software B.V. in Vught, the Netherlands. His research interest include: embedded systems design, advanced computer architectures, hardware/software co-design, VLSI design, cryptographic systems and computer systems testing. Contact him at Computer Engineering Laboratory, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands; georgi@dutepp0.et.tudelft.nl; <http://ce.et.tudelft.nl/~georgi/>.

Bruno Crispo received an M.Sc. in computer science from the University of Torino, Italy and a Ph.D. in computer science from the University of Cambridge, UK. He is currently an Assistant Professor of Computer Science at the Vrije Universiteit in Amsterdam. His research interests are security protocols, authentication, authorization and accountability in distributed systems and ubiquitous systems, sensors security. He has published several papers on these topics in referred journals and in the proceedings of international conferences. Contact him at Dept. of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands; crispo@cs.vu.nl; <http://www.cs.vu.nl/~crispo>.

Rutger F. H. Hofman has an M.Sc. in Physics and a Ph.D. in Computer Science from the Vrije Universiteit Amsterdam. He is a research programmer in the Computer Systems group at the Vrije Universiteit. Rutger has done lots of development and performance tuning for parallel and distributed systems, and his current work focus is on security-oriented systems. Contact him at Dept. of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands; rutger@cs.vu.nl; <http://www.cs.vu.nl/~rutger>.

Andrew S. Tanenbaum has an S.B. from M.I.T. and a Ph.D. from the University of California at Berkeley. He is currently a Professor of Computer Science at the Vrije Universiteit in Amsterdam. His research interests are reliability and security in operating systems, distributed systems, and ubiquitous systems. He is the author of five books that have been translated in 20 languages, as well as the author of over 100 published papers. He has lectured in over a dozen countries. Tanenbaum is a Fellow of the IEEE, a Fellow of the ACM, and a member of the Royal Dutch Academy of Sciences. Contact him at Dept. of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands; ast@cs.vu.nl; <http://www.cs.vu.nl/~ast>.

Bibliography

- [1] *Hold off on that chip, says thompson*, http://worldnetdaily.com/news/article.asp?ARTICLE_ID=47853.
- [2] ISO/IEC FDIS 15693, *Identification cards – contactless integrated circuit(s) cards – vicinity cards*, 2001.
- [3] ECMA-340, *Near field communication interface and protocol (nfcip-1)*, Dec., 2004.
- [4] EPCglobal, *13.56 MHz ISM band class 1 radio frequency (RF) identification tag interface specification*.
- [5] Feldhofer, Martin, Sandra Dominikus, and Johannes Wolkerstorfer, “Strong authentication for RFID systems using the AES algorithm,” *Workshop on Cryptographic Hardware and Embedded Systems*, LNCS, Vol. 3156, Aug., 2004, pp. 357-370.
- [6] Finkenzeller, Klaus, *RFID Handbook: Fundamentals and applications in contactless smart cards and identification*, John Wiley & Sons, Ltd., 2003.
- [7] FoeBuD, *Data privatizer*, Jul., 2005, https://shop.foebud.org/product_info.php/cPath/30/products_id/88.
- [8] Golle, P., M. Jakobsson, A. Juels, and P. Syver-son, “Universal re-encryption for mixnets,” *Proceedings of the 2004 RSA Conference*, 2004.
- [9] Großschädl, Johann and Stefan Tillich, “Design of instruction set extensions and functional units for energy-efficient public-key cryptography,” *Workshop on RFID and Lightweight Crypto*, Jul., 2005.
- [10] Juels, Ari, “Minimalist cryptography for low-cost RFID tags,” *The Fourth International Conf. on Security in Communication Networks*, LNCS, Springer-Verlag, September 2004.
- [11] Juels, Ari, Ronald L. Rivest, and Michael Szydlo, “The blocker tag: Selective blocking of RFID tags for consumer privacy,” *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ACM Press, 2003.
- [12] Juels, Ari, Paul Syver-son, and Dan Bailey, “High-power proxies for enhancing RFID privacy and utility,” *Proc. of the 5th Workshop on Privacy Enhancing Technologies*, 2005.
- [13] Karjoth, Günter and Paul Moskowitz, “Disabling RFID tags with visible confirmation: Clipped tags are silenced,” *Workshop on Privacy in the Electronic Society*, Nov., 2005.
- [14] Lingle, Rick, “MIT’s economical RFID field probe,” *Packaging World*, 2005.
- [15] Melexis, *Application Note: A power booster for MLX90121*, 001 ed., Apr., 2004, <http://www.melexis.com>.
- [16] Melexis, *MLX90121: 13.56 MHz RFID transceiver*, 006 ed., Dec., 2005, <http://www.melexis.com>.
- [17] Minime and Mahajivana, “RFID Zapper,” *22nd Chaos Communication Congress (22C3)*, Dec., 2005.
- [18] Rieback, Melanie R., Bruno Crispo, and Andrew S. Tanenbaum, “Keep on blockin’ in the free world: Personal access control for low-cost RFID tags,” *Proc. 13th Cambridge Workshop on Security Protocols*, Apr., 2005.
- [19] ———, “RFID guardian: A battery-powered mobile device for RFID privacy management,” *Proc. 10th Australasian Conf. on Information Security and Privacy (ACISP 2005)*, LNCS, Vol. 3574, Springer-Verlag, pp. 184-194, July, 2005.
- [20] Spiekermann, Sarah and Oliver Berthold, “Maintaining privacy in RFID enabled environments – proposal for a disable-model,” *Workshop on*

Security and Privacy, Conf. on Pervasive Computing, Apr., 2004.

- [21] Vajda, István and Levente Buttyán, “Lightweight authentication protocols for low-cost RFID tags,” *2nd Workshop on Security in Ubiquitous Computing*, Oct., 2003.
- [22] Weis, Stephen, Sanjay Sarma, Ronald Rivest, and Daniel Engels, “Security and privacy aspects of low-cost radio frequency identification systems,” *Security in Pervasive Computing*, LNCS, Vol. 2802, pp. 201-212, 2004.
- [23] Westhues, Jonathan, *For anything: proxmarkii*, Dec., 2005, <http://cq.cx/proxmarkii.pl>.

Specification-Enhanced Policies for Automated Management of Changes in IT Systems

Chetan Shankar – University of Illinois at Urbana-Champaign
Vanish Talwar, Subu Iyer, Yuan Chen, and Dejan Milojicić – Hewlett-Packard Laboratories
Roy Campbell – University of Illinois at Urbana-Champaign

ABSTRACT

Enterprise and grid computing systems are complex and subject to a broad range of changes such as configuration updates, failures, and performance degradations. These changes affect infrastructure elements such as computation and storage nodes, applications, and system management elements such as monitoring infrastructures. Today's best practices in use by system administrators to manage these changes are manual and ad-hoc. In large complex installations, this would lead to high operational costs, broken closed loop automation, and reduced agility. Providing tools and mechanisms to administrators that automate the reaction to these changes is highly desirable and is an active research area.

Policy-based management using Event-Condition-Action (ECA) rules is a well-known approach for such automated change management where management actions are executed when specified event-conditions are observed. In complex systems, the interdependence of components generates multiple events when a single change happens causing multiple rules to be triggered. The order of execution of rule actions determines the system behavior necessitating reasoning about execution order. ECA rules do not contain explicit action specifications needed for reasoning and are therefore unsuited for specifying management rules.

In this paper, we propose a specification-enhanced ECA model called Event-Condition-Precondition-Action-Postcondition (ECPAP) for designing adaptation rules. ECPAP rules contain action specifications in first order predicate logic enabling us to develop reasoning algorithms to determine enforcement order of multiple rules. The enforcement order is represented as a Boolean Interpreted Petri Net workflow. We introduce a new notion called enforcement semantics that provides guarantees about rule ordering. We have built an adaptation framework using ECPAP model and have demonstrated it for automated change management of Ganglia and HP OpenView monitoring systems. The evaluation of the framework illustrates the significance of the ECPAP model and demonstrates its applicability for managing complex IT environments.

Introduction

Modern IT systems, such as enterprise data centers and grids, are paradigms of distributed computing where computation and data are distributed across diverse computational and storage elements. These systems are characterized by growing complexity, scale, and heterogeneity of infrastructure and applications. Further, these systems are highly dynamic, and subject to frequent changes such as service plug-in/plug-out, workload variations, failures, configuration updates, and application migration. Such changes affect the runtime operation of the system, and the service contracts offered to customers. Therefore, in reaction to these changes, infrastructure elements, applications, as well as system management components need to be adapted. For example, compute and storage resources may have to be re-allocated, applications may need to be restarted, and monitoring infrastructures may require re-configuration.

Current approaches used by administrators to manage these changes are manual and/or a combination of ad-hoc tools and scripts. The process typically requires special expertise and detailed actions by administrators. While these approaches may work fine in small scale installations, they do not scale in larger scale installations typical of modern IT systems and utility systems of tomorrow. In such environments, there is a need to provide administrators with tools that can capture the expert domain knowledge in machine readable format and thereafter react to changes in an automated manner.

The use of Event-Condition-Action (ECA) rules [1] is a well-known approach for enabling system administrators to specify the desired actions to be invoked on changes in policy rules [5, 6, 7, 10, 20]. When a change event is received, the rules matching the event are determined. If the conditions in these rules are true, the corresponding actions are executed.

An example of an ECA rule is “When checkpoint store is full (*event*), if backup store is running (*condition*), assign backup store as new checkpoint store (*action*).” When the checkpoint store becomes full, an event is sent that triggers the rule. The management system verifies if the backup store is running and if so assigns it as the new checkpoint store.

Policy-based management systems have been effectively used to manage network switches [5], content distribution networks [6], and general distributed systems [7]. The applicability of policy-based systems for reacting to changes in today’s IT systems, such as data centers and utility infrastructures, presents numerous challenges due to highly interdependent components. Changes in these environments propagate rapidly causing several related changes.

Consider, for example, an infrastructure for monitoring performance of a data center as depicted in Figure 1. The infrastructure consists of a set of monitored nodes on which monitoring agents continuously collect performance data and send them to nodes called *aggregators*. These aggregators perform statistical functions on the collected data and send them to interested clients, such as archival stores and performance visualizers. An aggregator’s output typically represents the performance data of a cluster and is useful for viewing consolidated information.

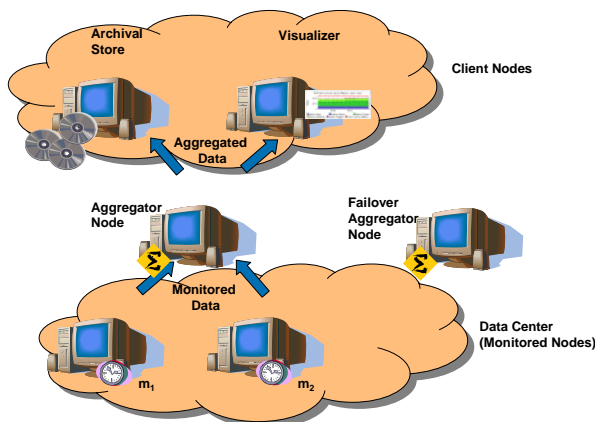


Figure 1: Infrastructure for monitoring performance of a data center.

If the aggregator node fails, monitored nodes cannot send data and may report failures. Similarly, clients using the performance data may report failures as they do not receive any data from the aggregator. Other clients such as load balancers and job schedulers using this data may unnecessarily rebalance the load or schedule jobs incorrectly thus affecting the overall utilization of the system.

Change detectors monitoring different aspects of the system sense several changes due to aggregator failure and generate various events. This causes several policy rules to get triggered in the adaptation system.

The order of enforcement of rules determines the final system state. As illustrated in Figure 2, the failure of aggregator node generates multiple events. These events trigger two rules – R_1 and R_2 . R_1 states: “If aggregator node fails, assign failover node as aggregator” and rule R_2 states: “If data-send from monitoring agent fails, reconnect to aggregator.” When the aggregator node fails, both rules are triggered. If R_1 is enforced before R_2 , the failover node is assigned as aggregator and the monitoring agent is reconnected to it. But if R_2 is enforced before R_1 , the action of R_2 fails since the monitoring agent tries to reconnect to the stopped aggregator.

Therefore, the order of enforcement of rules determines the final system state when multiple rules are simultaneously triggered. An adaptation system should reason about the enforcement order of rules and provide guarantees for system behavior to be deterministic.

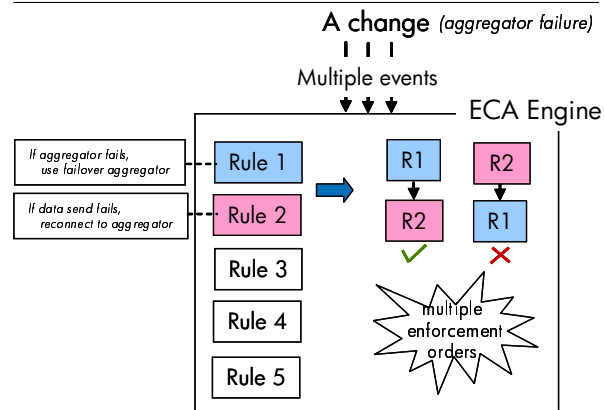


Figure 2: Problem with ECA based system.

ECA rules do not contain explicit action specifications needed for reasoning and are therefore unsuited for specifying management rules in such environments. Therefore, we propose an extended model of ECA called Event-Condition-Precondition-Action-Postcondition (ECPAP) for designing management rules. These rules contain the axiomatic specification of rule actions in first-order predicate logic as pre- and post-conditions. The pre-condition specifies the partial system state before execution of rule action while post-condition specifies the partial system state once the action has successfully executed.

Note that the rule condition is different from precondition because the rule condition is specified by the policy designer while the pre-condition is specified by the action developer (programmer). We have used the ECPAP rule framework for conflict detection and resolution and monitoring of rule enforcement in [11] and analyzing policy cycles in [10]. In this paper, we show how the pre- and post-conditions can be used to determine the dependencies among triggered rule actions and reason about their enforcement order.

Given a set of triggered rules, the adaptation system dynamically generates a Petri net workflow representing dependencies among rule actions. An action

depends on another action if the pre-condition of the former is satisfied by the post-condition of the latter. Since specifications are in first-order predicate logic, which are undecidable in the general case, dependencies can only be determined at runtime when the specifications are instantiated to form propositional expressions. Therefore, workflows of rule actions cannot be constructed statically and must be done at runtime.

Certain actions may be independent of other triggered actions and therefore may not be present in the workflow. In such cases, the system could take one of several possible decisions, for, e.g., it may abort the workflow, or it may execute the maximum possible actions and so on. We identify the need for policy-based systems to provide semantic guarantees for rule enforcement in such circumstances.

We introduce a notion called *rule enforcement semantics* for policy-based systems that provides certain guarantees when multiple rules are concurrently triggered. We have identified three enforcement semantics – random, all-or-none and maximum rule – that have been found useful in different circumstances. We discuss this in detail in section “Ordering Rule Enforcement.”

We make the following contributions in this paper:

- We propose a specification-enhanced rule framework called Event-Condition-Precondition-Action-Postcondition (ECPAP) for specifying adaptation rules.
- We present algorithms to determine enforcement order of multiple rules using the ECPAP model. The enforcement order is represented as a Boolean Interpreted Petri Net (BIPN) workflow. We also introduce a new notion called enforcement semantics that provides guarantees about rule enforcement.
- We describe an adaptation framework built using the ECPAP model and demonstrate its application for automated change management of Ganglia and HP OpenView monitoring systems.
- We present evaluation results that illustrate the need for enforcement guarantees and the feasibility of the ECPAP model.

Our extension of the ECA framework with action specifications follows naturally from current research efforts in autonomic computing. There has been widespread interest lately on using planning techniques from AI for programming and managing distributed systems with encouraging results [10, 11, 12, 13, 24]. In [10, 11] we showed how extending actions with specifications enabled advanced conflict and termination analysis for policy-based management systems. Andrzejak, et al. [12] have used actions with pre- and post-conditions for planning complex workflows from simple actions for system management.

The ABLE project [13] uses axiomatic specifications of actions for goal-based autonomic computing.

Anand, et al. [24] use specification-enhanced actions, expressed as pre-conditions and effects, for programming pervasive computing environments. These research works have shown that annotating actions with simple pre- and post-condition specifications provides numerous benefits such as raising the programming abstraction level and automating system management. Based on the success of these efforts we have extended management policies with action specifications and introduced the ECPAP rule framework.

The rest of the paper is organized as follows. In the next section, we present in detail the ECPAP rule framework. We then describe the workflow generation algorithms and enforcement semantics. Subsequently, we present the ECPAP-based adaptation framework and its application for managing monitoring systems. We then discuss evaluation results and some lessons learned. Finally, we present related work and a conclusion.

Specification-Enhanced Rule Framework

The ECA rule framework is used in different paradigms such as active databases, access control and system management to react to different situations. Active databases use the ECA framework for designing triggers that specify actions to be executed when certain database operations such as record insertion or deletion are made. Access control systems use ECA rules to authorize or deny access when an access request is made. Management systems use the ECA framework for designing *obligation rules* [7] to specify management actions to be executed when system changes are observed. Rule actions in active databases and access control are normally well-defined and hence their effects on the system are implicitly known.

For example, active database trigger rules normally use insert, delete and update actions [21] while access control actions are normally authorize, deny and delegate. This enables complex reasoning such as confluence [21], rights-amplification and conflict analyses to be performed over these rules. But rule actions in system management are not well-defined and can range from simple atomic actions to complex scripts and so their effects on the system are not implicitly known. Therefore, explicitly specifying the action effects using pre- and post-conditions enables complex reasoning to be performed over management rules. This motivated us to design the ECPAP rule framework.

The ECPAP framework extends the ECA framework by using the Hoare triple [23]. A Hoare triple represented as $\{P\}C\{Q\}$ describes how an action C changes the state of computation from a state where P is true to a state where Q is true. P and Q , expressed as first-order predicate logic expressions, are pre- and post-conditions of C , respectively and are called axiomatic specifications. The pre-condition specifies the system state that should exist before C can be executed.

Our adaptation policies are formulated as sets of ECPAP rules of the form

on event if condition do action

A policy rule is read as: “When event occurs in a situation where condition is true, then execute action.” The action is a call to a method in a library of actions where each action is annotated with a pre-condition and a post-condition by the action developer (programmer). Note that pre- and post-conditions are not specified as part of the rules since an action may be invoked by multiple rules in the policy and this format avoids listing the specifications at multiple places.

We represent an ECPAP rule as $(e, c, p) \rightarrow (a, s)$ where e denotes the rule event, c denotes the condition of the rule, p is the pre-condition of the action, a is the action to be executed and s is the action post-condition. Our policy rule framework extends that of Policy Description Language (PDL) [1] by adding axiomatic specifications as “extension”s to the rule.

Policy Syntax and Semantics

There are three basic classes of symbols: primitive event symbols, action symbols and constant symbols. Primitive event symbols represent basic events that can be subscribed to in the system. For example, *MonNodeFail* and *ServiceFail* are primitive event symbols that are generated when a monitored node or service fails. An event is a primitive event symbol or a term of the form $e(T_1 t_1, \dots, T_n t_n)$, where e is a primitive event symbol of n arguments and each t_i is a constant or a variable of type T_i . $e(T_1 t_1, \dots, T_n t_n)$ represents a parameterized event where the parameters are bound to the data contained in the event.

The condition part of an ECPAP rule is a boolean expression containing constants and variables that appear in the event part of the rule.

Each action symbol denotes the name of a procedure that can be invoked in the system. An action is of

the form $proc(t_1, \dots, t_n)$ where $proc$ is an action symbol and t_i s are parameters. For example, *startService(S)* is an action. Actions are defined in an action library that also contains pre- and post-conditions of actions.

Pre- and post-conditions of an action are first-order predicate logic formulas of the form

$$p_1 (\wedge \mid \vee p_k)^{k=2..m},$$

p_i is a first-order predicate of the form $Q_i t_i \in X_i, \dots, Q_n t_n \in X_n$ $pred(t_1, \dots, t_n)$: Q_i is a quantifier, X_i is a constant symbol and each t_i is a constant or a variable.

A policy, P is a finite set of ECPAP rules. The adaptation system enforcing the policy expects as input an event e , and its occurrence is represented by $occ(e)$. The semantics of each rule, $(e, c, p) \rightarrow (a, s)$ in the policy is specified by the implication,

$$\begin{aligned} occ(e) \wedge c \wedge p &\rightarrow exec(a) \\ exec(a) &\rightarrow s \end{aligned}$$

where $exec(a)$ represents the initiation of the execution of action a . The evaluation of the rule and execution of the action is treated as an atomic operation, i.e., if the system state changes after the rule evaluation and before the action execution, the change is ignored.

Pre- and Post-condition Expressions

The pre- and post-conditions use pre-defined keywords for specifying first-order expressions. For example, *MonitoredNodes* and *ClientNodes* are keywords that represent sets of monitored nodes and client nodes in our system. In our example scenario these sets are: $MonitoredNodes = \{m_1, m_2\}$ and $ClientNodes = \{store, visual\}$. A quantifier over these sets enumerates all the elements of the set. First-order expressions are undecidable in the general case and therefore we convert them to propositional logic expressions, which are decidable, during evaluation.

```

R1: on(AggregatorFail(Node n))
    if(n.id != "FailOver")
    {statusNode("FailOver", running)}
    do (UseNodeAsAggregator("FailOver"));
    {statusAggregator(running)}

R2: on(DataSendFail(Node n))
    if(n.id == "MonitoredNode")
    {statusAggregator(running)}
    do(ReconnectToAggregator(n));
    {connectionStatusToAgg(n, connected)}

R3: on(DataReceptionFail(Node n))
    if(true)
    {statusAggregator(running)}
    do(ReconnectToAggregatorAsClient(n));
    {connectionStatusToAgg(n, connected)}

R4: on(AggregationAgentStopped(Node n))
    if(true)
    {statusAggregator(running)  $\wedge \forall x \in MonitoredNodes, connectionStatusToAgg(x, connected) \wedge$ 
       $\forall x \in ClientNodes, connectionStatusToAgg(x, connected)}$ 
    do(RestartAggregationAgent());
    {statusService("AggregationAgent", running)}

```

Policy 1: Adaptation Policy for Performance Monitoring Scenario.

Consider, for example, $\forall x \in \text{MonitoredNodes}, \text{statusNode}(x, \text{running})$ is a first-order expression that is converted to $\text{statusNode}(m_1, \text{running}) \wedge \text{statusNode}(m_2, \text{running})$ before analysis. These expressions are evaluated by invoking corresponding methods in the system and verifying the return values. For example, $\text{statusNode}(m_1, \text{running})$ is evaluated by comparing the return value of the call $\text{statusNode}(m_1)$ with the string “running.”

Example Adaptation Policy

The adaptation policy for the scenario described in the first section is shown in Policy 1. The pre- and post-conditions of actions are shown italicized in braces for convenience and are not specified as part of the rules.

Rule R_1 gets triggered when the aggregator node fails and if the failed node is not the failover node, it assigns the failover node as the new aggregator. Rule R_2 is triggered when any monitored node fails to send data to the aggregator. The rule tries to reconnect the monitored node to the aggregator. Rule R_3 is triggered when a node fails to receive data from the aggregator node. The rule reconnects the node to the aggregator node. Rule R_4 is triggered when the aggregation agent stops. The aggregation agent needs to be started everytime new clients or monitored nodes are connected to the aggregator node. This rule restarts the aggregation agent.

When the aggregator node fails, the monitored nodes are unable to send data and the archival store and visualizer nodes do not receive any data. Therefore, one *AggregatorFail* event, two *DataSendFail* events (one each from two monitored nodes), two *DataReceptionFail* events (one each from the archival store and visualizer nodes) and one *AggregationAgentStopped* events are generated. These events trigger multiple instances of the above rules.

If both instances of rules R_2 and R_3 are enforced before R_1 , the nodes try to connect to the failed aggregator and therefore do not succeed. If R_4 is enforced before the other rules, the aggregation agent restart fails and so the nodes can neither send nor receive data. But if R_1 is enforced before R_2 and R_3 and R_4 is enforced in the end, the nodes get connected to the failover aggregator and the monitoring activity is restored. Therefore, the order of enforcement of rules determines system behavior. While “correct” order of rule enforcement is hard to define and requires experimental justification, other simpler guarantees about ordering can be provided as will be discussed shortly.

Ordering Rule Enforcement

An adaptation policy is subject to numerous changes such as addition and deletion of rules, rule modifications and policy composition. Each rule is generally evaluated and enforced independent of other rules in the policy. When multiple rules are triggered the order of enforcement of rules determines the system

behavior, as demonstrated in the previous section. Therefore, we define a new notion called *enforcement semantics* that provides certain guarantees about rule enforcement. Enforcement semantics of a policy-based adaptation system dictates the way rules are to be enforced when multiple rules are simultaneously triggered.

V	: set of trivially-enabled actions
A	: set of actions of triggered rules
$\text{Enable}(a)$: set of actions enabled by action a
$P = \{\text{Start}\}$: set of Petri net Places – initialized to Place called ‘Start’
$T = \{\}$: set of Petri net Transitions
$\text{place}(a)$: Place for action a
$\text{adj}(x)$: adjacency list of x represented as a set, where $x \in P \cup T$
$\text{trans}(p, f)$: Transition with Boolean function f connected by edges from places in set p

Figure 3: Notations used in the workflow generation algorithms.

When a set of rules is triggered, the execution order of the rule actions is determined by constructing a workflow that expresses dependencies between different actions. The pre- and post-conditions of actions determine which action enables which other actions. An action is said to *enable* another action if the post-condition of the former satisfies the pre-condition of the latter. In our example scenario, the post-condition in rule R_1 satisfies the pre-condition in R_2 . Therefore, action of R_1 is said to enable that of R_2 .

The workflow of rule actions is represented as a Boolean Interpreted Petri net (BIPN) [4], which is useful to model and reason about concurrent action execution. A Boolean Interpreted Petri net is a Petri net [3] whose transitions are assigned Boolean functions. A transition can fire only when all of its input places are marked and its Boolean function is *true*. We assign a place to each action and each transition leading to the place is assigned the pre-condition of the action as the Boolean function. We formally define our workflow BIPN in Appendix I. We will describe the algorithms that construct the workflow in the rest of this section. The various notations used in the algorithms in this section are catalogued in Figure 3.

Workflow Construction

The workflow is constructed by analyzing each pair of actions to determine if one enables the other. The current system state can be represented as a set of propositions and pre-conditions of certain actions may be satisfied by it. These actions are independent of other triggered rules and can be executed as the first set of actions in the workflow. These actions are called trivially-enabled actions.

Definition 1: An action a is said to be *trivially-enabled* if the current state of the system, I , satisfies its pre-condition. Formally, it is represented as $I \models \text{pre}(a)$, where \models is the *satisfies* symbol.

In our example scenario, the pre-condition in R_1 : *statusNode("FailOver", running)* is satisfied by the current system state since the failover system is running when the aggregator node fails. Therefore, the action of R_1 can be executed independent of actions in other triggered rules. Algorithm 1 to determine trivially-enabled actions is shown in Figure 4a.

The algorithm initializes the Petri net by assigning a place to each action and creating a transition with the Boolean function *true*. This transition is connected to the *Start* place. The algorithm evaluates the pre-condition of each action to determine if it is true and marks the action as trivially-enabled if so. These trivially-enabled actions are connected by edges from the *true* transition. In our adaptation scenario, only action A_1 is trivially-enabled (action of rule R_i is represented as A_i). The Petri net workflow that results from the algorithm for our adaptation scenario is shown in Figure 4b.

Once trivially-enabled actions have been identified, we check to see which action enables which other actions through *enablement analysis*.

Definition 2: An action a_1 is said to *enable* action a_2 if $post(a_1) \models pre(a_2)$ where $post(a_1)$ represents the post-condition of action a_1 and a_2 is not trivially-enabled.

This implies that execution of a_1 would satisfy the pre-condition of a_2 and so a_2 can be executed after a_1 . Since any proposition satisfies the true proposition, we do not check if post-condition of an action satisfies pre-condition of a trivially-enabled action.

Algorithm 2 for enablement analysis is shown in Figure 5a. This algorithm verifies for each triggered action if its post-condition satisfies the pre-condition of a non-trivially-enabled action. It does a pair-wise satisfiability check of actions to determine enablement. The set $Enable(a)$ contains all actions that are enabled by action a . The algorithm iterates through each action a and if a enables other actions, it connects them to a through transitions labeled with their pre-conditions.

In our example scenario, both instances of action A_2 (corresponding to two monitored nodes) and both instances of action A_3 (corresponding to archival store and visualizer) are enabled by A_1 . The Petri net resulting from Algorithm 2 is shown in Figure 5b. A_k^i represents the i th instance of action A_k . Since two instances of rules R_2 and R_3 are triggered, the Petri net contains two instances of their actions represented as A_2^i and A_3^i , ($i = 1, 2$).

Post-conditions of some actions may satisfy part of the pre-condition of another action. For example, post-condition of A_1 : *statusAggregator(running)* satisfies a part of the pre-condition of A_4 . Similarly, post-conditions of A_2^1 , A_2^2 , A_3^1 and A_3^2 satisfy the other parts of the pre-condition of A_4 . Therefore, A_1 , A_2^i and A_3^i must be executed to enable A_4 . We say that each

```

for each action a ∈ A //initialization
  P = P ∪ {place(a)}
  t = trans({start}, true)
  adj(Start) = adj(Start) ∪ {t}
  T = T ∪ {t}
V = {} //trivially-enabled action analysis
for each action a in A
  if pre(a) is true
    V = V ∪ a
for each action a ∈ V //adding transitions to workflow
  adj(t) = adj(t) ∪ {place(a)}

```

Figure 4a: Algorithm 1: Workflow Initialization and Trivially-enabled action analysis.

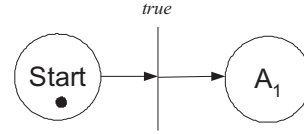


Figure 4b: Petri net workflow after trivially-enabled action analysis.

```

Enable(a) = {}, ∀ a ∈ A //enablement analysis
for each action a ∈ A
  for each action b ∈ A-V
    if post(a) ⊨ pre(b)
      Enable(a) = Enable(a) ∪ {b}
for each action a ∈ A //adding transitions to workflow
  for each action b ∈ Enable(a)
    t = trans({place(a)}, pre(b))
    if t ∉ T
      T = T ∪ {t}
      adj(place(a)) = adj(place(a)) ∪ {t}
    end if
    adj(t) = adj(t) ∪ {place(b)}
  end for
end for

```

Figure 5a: Algorithm 2: Enablement Analysis.

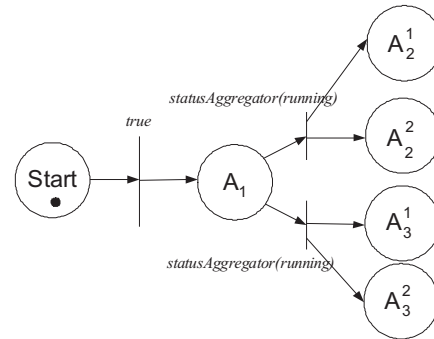


Figure 5b: Petri net workflow after enablement analysis.

action A_1 , A_2^i and A_3^i partially-enables A_4 . Note that the variables x and n in predicates *connectionStatusToAggregator(x, connected)* and *connectionStatusToAggregator(n, connected)* are bound to values of the nodes during evaluation.

Definition 3: An action a_1 is said to partially-enable action a_2 if $\text{post}(a_1) \models \text{partial-pre}(a_2)$, where $\text{partial-pre}(a_2)$ is a conjunction of some proper subset of conjuncts of $\text{pre}(a_2)$. A set of partially-enabling actions of an action a that together enable a is called a *partial-set* of a . An action may have multiple partial-sets and therefore, the set of all partial-sets of a is denoted by $\text{partial-sets}(a)$. In the above example,
 $\text{partial-sets}(A_3) = \{ \{A_1, A_2^1, A_2^2, A_3^1, A_3^2\} \}$.

Algorithm 3 in Figure 6a determines for every action a that is not trivially-enabled, which set of actions collectively enable a . If the set contains only one action, then it implies that a single action enables a and therefore is already determined by Algorithm 2. Therefore, Algorithm 3 only considers sets having more than one element. In addition, the algorithm does not test an action with itself for partial-enablement as this might lead to a deadlock.

Though the algorithm for partial-enablement analysis can replace enablement analysis of Algorithm 2, we separate the two algorithms since partial-enablement analysis has a much higher complexity. We will discuss this in more detail below when we evaluate the algorithmic complexities.

Once we determine the partial-sets, we complete the workflow construction by adding transitions. The Petri net generated from Algorithm 3 is shown in Figure 6b.

Enforcement Semantics

Once dependencies among triggered rule actions have been determined, the enforcement semantics of the adaptation system specifies the execution order of actions. We have identified three different enforcement semantics for policy-based adaptation systems.

Random

This semantics executes rule actions in a random order. The pure ECA policy system without the specification enhancements follows this semantics, implicitly, since it does not provide guarantees about enforcement of multiple triggered rules. This is the weakest of all

```

Partial-sets(a) = {}
S      : set that temporarily contains partially-enabling
         actions of an action
for each action a ∈ A-V      //partial sets determination
  S = {}
  for each action b ∈ A-{a}
    if b partially-enables a
      S = S ∪ {b}
  for each subset s of S
    if (cardinality(s) > 1)
      p = true
      for each action a ∈ s
        p = p ∧ post(a)
      if p satisfies pre(a)
        Partial-sets(a) = Partial-sets(a) ∪ {s}
    endif
  end for
end for

for each action a ∈ A-V      //adding transitions to workflow
  for each set s ∈ Partial-sets(a)
    t = trans(s, pre(a))
    T = T ∪ {t}
    adj(t) = adj(t) ∪ {place(a)}
    for each action b ∈ s
      adj(place(b)) = adj(place(b)) ∪ {t}
    end for
  end for
end for

```

Figure 6a: Algorithm 3: Partial-sets Determination.

three semantics and does not require the action workflow to be constructed. This semantics can be used when dependency among rule actions is low and very few rules are triggered by a single change.

All-or-None

The all-or-none semantics specifies that the rule actions in the workflow must be executed only if all actions can eventually execute. This implies that even if one action in the workflow cannot be enabled then the entire workflow should be discarded. In order to

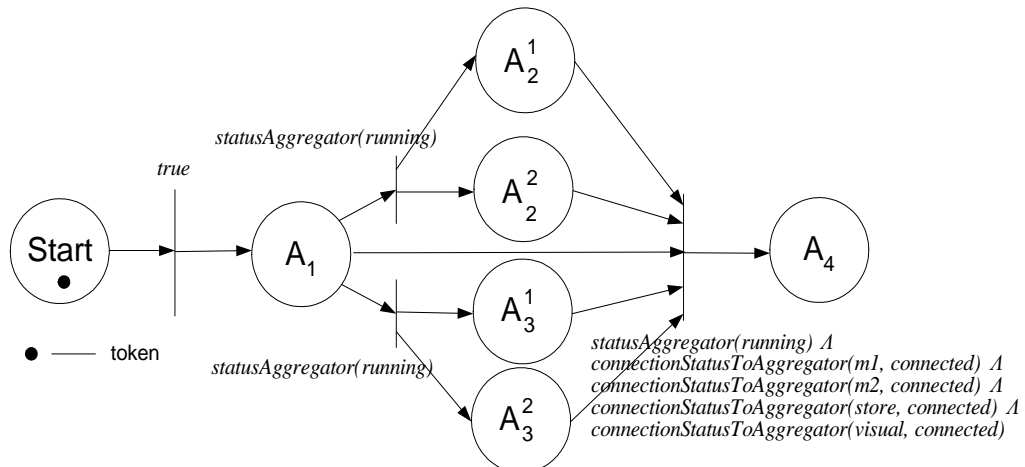


Figure 6b: Final Petri net workflow.

enforce this semantics, the BIPN workflow is analyzed to see if all places can be reached using a reachability algorithm [3]. The all-or-none semantics provides the strongest guarantee and is useful in policies that have high dependency among rule actions.

Maximum Rule

The maximum rule semantics guarantees that the management system enforces rules in an order that ensures as many rules are successfully enforced as possible, provided no other errors cause rule enforcement to fail. The difference between all-or-none and maximum rule enforcement semantics is that in the latter if any place in the workflow can be reached from the *Start* place it will be executed. If a place cannot be reached, the workflow is not discarded as in the all-or-none semantics. Our adaptation framework discussed in the “Framework” section uses the maximum rule enforcement semantics.

We prove formally in Appendix I that the workflow algorithms described above guarantee the above semantics by showing that ordering actions according to the workflow enables maximum number of rules to be successfully enforced.

Action Execution

The order of execution of rule action depends on the enforcement semantics used in the system. If random enforcement is used, the workflow construction is skipped and actions are executed in an arbitrary order. The all-or-none and maximum rule enforcement semantics use Petri net based traversal algorithms to traverse the workflow and execute actions. If the system guarantees all-or-none semantics a reachability analysis [3] is performed to determine if all places are reachable from the *Start* place prior to execution.

A workflow execution engine analyzes the Petri net for any deadlocks using the deadlock detection algorithm described in [3]. If a deadlock is found the execution engine does not execute any action in the workflow. Currently, we do not resolve deadlocks and abandon the workflow. If the Petri net is deadlock-free, the engine uses a simple Petri net traversal algorithm based on Breadth-First Search (BFS) to traverse the net and execute actions.

The transition states of the Petri net act as synchronization points in the workflow. When multiple places lead to a single transition, the engine waits for the completion of all actions in the places before executing actions of places leading out of the transition. At each transition, the engine verifies the Boolean function for satisfaction before executing the following action. For our adaptation scenario, action A_1 is executed, followed by concurrent execution of actions A_2^i and A_3^i and then action A_4 is executed.

ECPAP-based Adaptation Framework

We have designed a framework based on ECPAP rules for adapting Ganglia and HP OpenView monitoring systems to various changes. The framework is

external to the monitoring systems and does not modify either system. It uses the reconfiguration support provided by the systems for adaptation. In this section, we discuss the details of the framework and its applications.

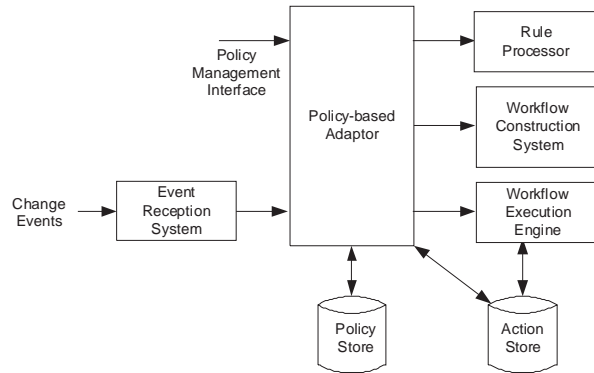


Figure 7a: Adaptation Framework.

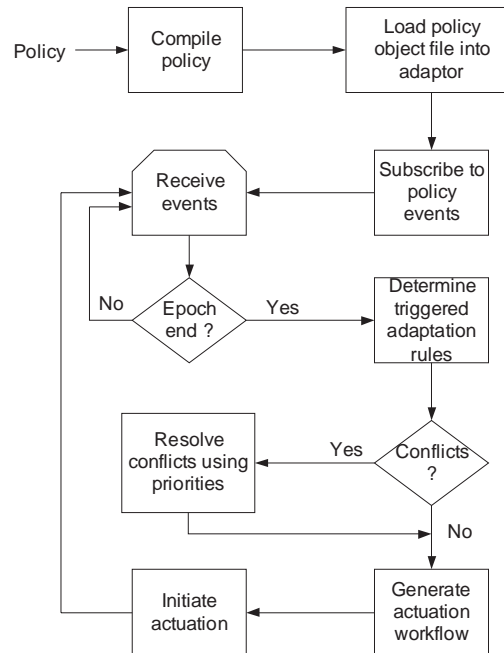


Figure 7b: Adaptation flowchart.

System Architecture

Adaptation policy, containing ECPAP rules, is compiled and loaded into the adaptation system by the system administrator. The adaptation system subscribes to events specified in the policy and initiates corresponding actions when those events are fired. Figure 7a shows the adaptation framework and Figure 7b illustrates the steps involved in loading a policy, planning adaptation and initiating actions.

A policy is compiled into a policy object file. The policy object file contains rules in a format suitable for loading into the enforcement system. The enforcement system subscribes to policy events and waits for the occurrence of events

Since a single change to the monitoring system may trigger more than one event occurrence, we define time intervals called *epochs* and consider all events received within an epoch to correspond to a single change. The epoch model for event reception was proposed in [2] and was found suitable for defining policy rules with composed events.

Since a composed event normally contains events that have occurred “simultaneously” and event reception system receives events sequentially, the epoch model provides a good approximation to simultaneity. At the end of each epoch, the adaptor evaluates the policy and determines the set of rules that are triggered. The triggered rules are checked for conflicts and resolved using a priority-based resolution technique [10]. The adaptor reasons about the enforcement order of rules using pre- and post-conditions of actions and generates the Petri net workflow. The workflow is executed by a workflow execution engine and the adaptor waits for further events.

The policy-based adaptor supports interfaces to load policies and conflict resolution rules, query the policy store for the loaded policies and retrieve the set of actions in the action store. In addition, the system also supports user interfaces to list available events and actions. These interfaces are useful for designing policies.

Event Reception Model

Our monitoring framework views a *change* as a set of correlated events and evaluates the policy based on the events in the set. Event correlation is a well-researched problem and numerous models have been proposed to group events corresponding to a change [25, 26, 27]. Since the focus of our work is on policy evaluation and enforcement, we use a simple event correlation model based on *epochs* proposed by Chomicki, et al. [2] for policy evaluation. Figure 8 illustrates the epoch model. The input to our adaptation framework is a set of events and therefore, the

epoch model can be replaced by more appropriate correlation models without affecting policy evaluation and enforcement.

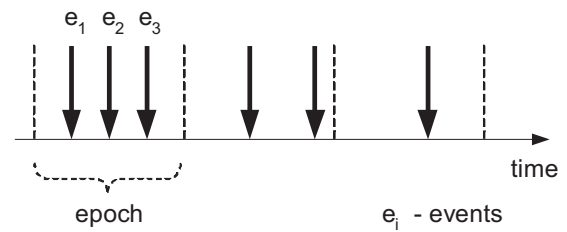


Figure 8: Epoch model.

Policy Tools and Adaptation Implementation

The adaptation framework consists of policy tools and the adaptation system. The policy compiler generates Java class files from policy rules. We used this approach to leverage the language features of Java. An ECPAP rule is compiled into a method that has the same signature as the event. The condition part of the rule is translated into an ‘if’ block in the method. The conditional expression of the ‘if’ statement is the same as that in the rule. The rule action is translated into a set of statements that create a Java object containing the action object along with its pre- and post-conditions. If multiple rules have the same event signatures, the compiler consolidates the rules into a single method with multiple ‘if’ blocks. A typical Java class, for a rule, generated by the policy compiler is shown in Figure 9.

These classes are compiled by a Java compiler into class files and loaded into the adaptation system by the policy loader. When an event is received by the adaptation system, an equivalent method signature is created with the event name as the method name and the event parameters as method parameters. This method is invoked on the rule class files and if the rule contains the event, the method invocation succeeds.

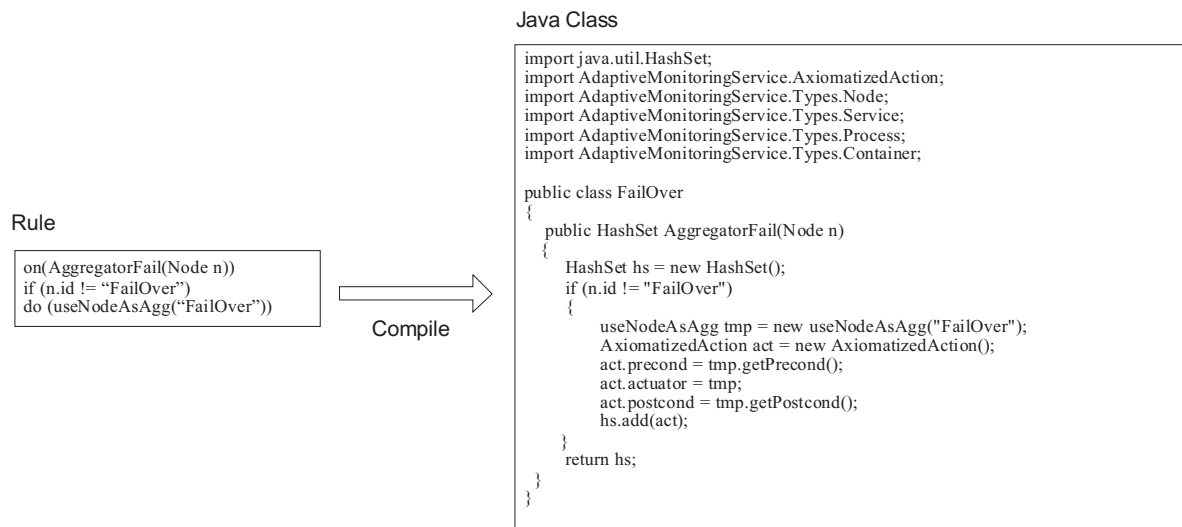


Figure 9: Policy compilation.

The condition is checked and if it is satisfied, an object containing the action object along with its pre- and post-conditions is returned. This approach provides the adaptation system with the action specifications necessary for reasoning.

The adaptation system has been implemented in Java. Figure 10 shows the main components of the framework. It consists of a policy store that stores a set of Java objects instantiated from rule classes. An action library acts as an actuator store and contains a set of actions that can be invoked from policy rules. The action library in addition contains specification of actions in first order predicate logic.

Application to Monitoring Systems

Our data center and enterprise systems use HP OpenView and Ganglia systems for performance monitoring. Since monitoring systems run independent of the core system services, we tested the applicability of our adaptation framework to manage changes to these systems.

The configuration of our monitored environment consists of a set of nodes monitored by the Ganglia monitoring system [14]. Ganglia is a scalable distributed monitoring system for high-performance computing systems such as clusters and grids. It supports a hierarchical organization of monitoring agents called Ganglia Monitoring Daemons (gmond) that collect monitoring information from individual nodes.

Aggregation agents called Ganglia Meta Daemons (gmetad) collect data from gmonds and apply aggregation functions to provide consolidated information about cluster of machines. The focus of Ganglia is on monitoring nodes and provides simple replication-based approaches for tolerating node failures. It does not provide support for aggregation agent failures, application migration or other infrastructure changes.

We used the adaptation framework to enhance the resilience of Ganglia monitoring system to changes as proof-of-concept. We developed change detectors to detect aggregator failures and data-send

and data-receive failures from the various nodes of the monitored environment. The change detection system generates parameterized events that contain relevant state information. The reasoning system uses XSB Prolog [19] to verify satisfiability of propositions.

We also used the adaptation framework for adapting OpenView monitoring system. The framework enables adaptation when services are plugged-in, plugged-out or migrated, alarm events are generated and so on. For example, the adaptive system dynamically configures the OpenView components to collect performance data from a service when it is plugged-in. Similarly, the framework reconfigures the components to collect additional metrics, through deep-diving (monitoring specific components), when alarms are generated due to high CPU utilization, low memory and so on. Our initial evaluation demonstrates that the ECPAP-based adaptation framework can dramatically reduce the administration cost of OpenView monitoring system.

Evaluation

In this section, we will discuss the algorithmic complexities of the various algorithms presented in the paper and use them to explain the system performance that we have empirically measured.

Algorithmic Complexity

Trivially-enabled action analysis (Algorithm 1) has a linear complexity of $O(n)$ pre-condition checks for n actions. Enablement analysis (Algorithm 2) does a pair-wise satisfiability check of actions and therefore has a quadratic complexity of $O(n^2)$. Partial-enablement analysis (Algorithm 3) analyzes for each action if it is enabled by a set of actions.

Each action subset must be determined and this has an exponential complexity of $O(2^n)$. Since each subset is tested to see if it enables the action for all actions the final complexity is $O(n^2 2^n)$. Currently, Algorithm 3 has a very high complexity but there are various optimizations that can be performed to reduce the value of n .

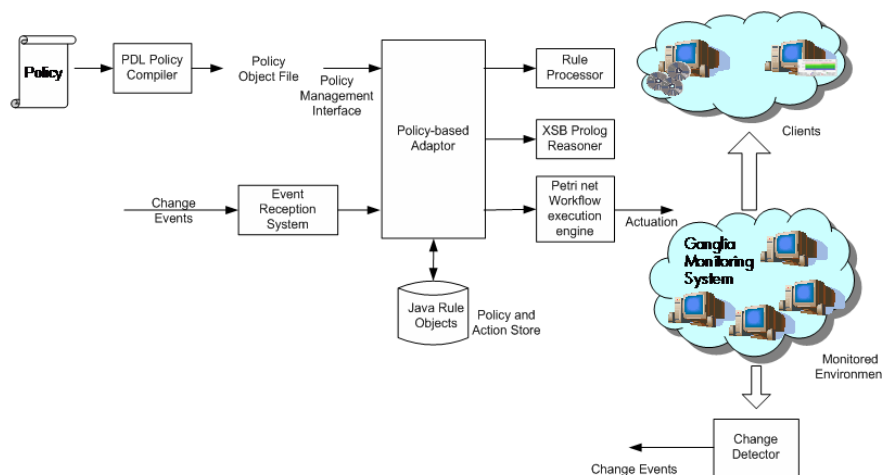


Figure 10: Adaptation framework implementation.

For example, the enablement analysis algorithm reduces the number of rules to be verified during partial-enablement analysis. Since enablement analysis has a quadratic complexity the overall performance overhead is greatly reduced. In addition, the number of rules that are normally triggered on a single event is quite less (fewer than five rules per event in our adaptation policy) and so the overhead is tolerable. We are currently looking at static analysis techniques to determine dependencies between different rules at policy compilation time. Finally, the overall complexity of the workflow generation is $O(n^2 2^n)$, bounded by the complexity of the partial-enablement analysis.

Experimental Validation

The performance overhead of Petri net workflow generation is shown in Figure 11. The adaptation system was executed on a Pentium III 1GHz dual processor SMP machine with 2 GB memory. Figure 11a shows the overhead with varying number of triggered rules. Our test policy had multiple instances of the same rule since the focus was on testing the overhead of the system. As predicted from the algorithmic complexity described above the overhead is exponential with the number of triggered rules. For 15 triggered rules the overhead was found to be around three seconds. Normally, for a typical policy, the number of rules triggered on a single change can be expected to be much less than 15 and so the approach is feasible. Several optimization using configuration and state models can be performed to reduce the overhead. We do not discuss these optimizations since they are out of the scope of the paper.

The number of predicates in pre- and post-conditions of actions influences the Petri net generation overhead. Therefore, we measured the overhead with varying number of predicates in action specifications. Figure 11b illustrates the performance overhead of the system. The x-axis indicates the average number of predicates for each pre- and post-condition. The y-axis shows the overhead in seconds. The overhead is less than one second for about 144 predicates.

Figure 12 shows the graphs of times required for policy compilation and rule evaluation. Policy compilation has a reasonable overhead of about one second for a policy containing 400 rules. Since policies are compiled only when they are modified, this is an acceptable overhead. Rule evaluation is an important component of the adaptation process and therefore its overhead contributes to the overall performance of the system. We measured the performance of the adaptation system with varying number of rules in a policy. We found that rule evaluation takes about 100 ms for a policy with 2000 rules which is a reasonable overhead for an adaptation system.

Experience in a Real Scenario

We have used our adaptation system for managing changes to monitoring systems as was described earlier. Figure 13 shows the Ganglia visualizer output

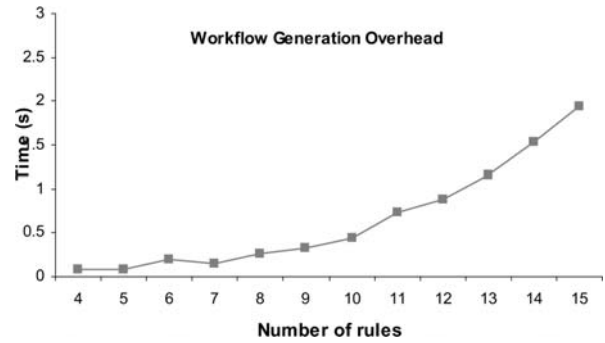


Figure 11a: Petri net Workflow Generation vs Number of Triggered Rules.

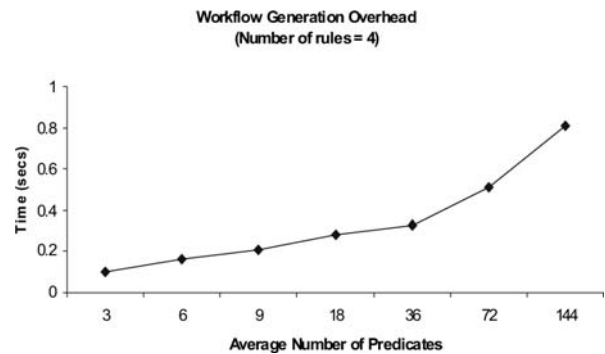


Figure 11b: Petri net Workflow Generation vs Average Number of Predicates.

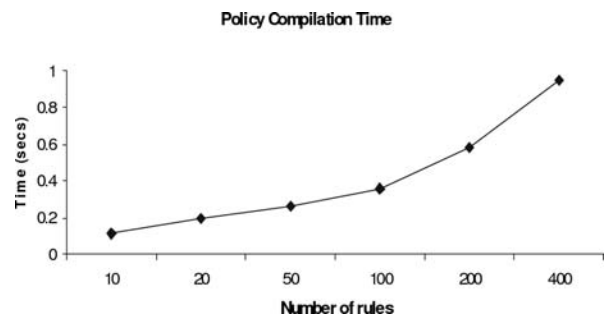


Figure 12a: Policy Compilation Times.

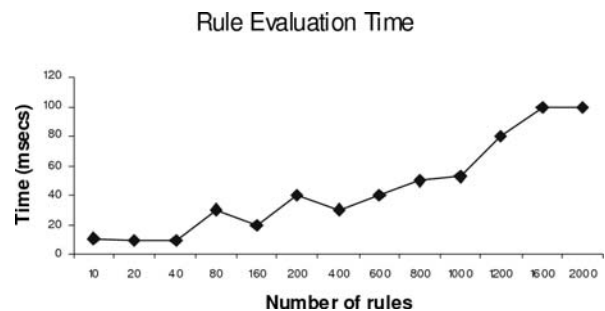


Figure 12b: Rule Evaluation Times.

on aggregator failure for our adaptation scenario. The policy consisted of four rules shown in the “Example Adaptation Policy” section. On aggregator failure, six events – one AggregatorFail, two DataSendFail, two

DataReceptionFail and one AggregationAgentStopped event – are generated. In the first experiment (Figure 13a), an ECPAP system with maximum rule enforcement enabled was used and the aggregator was manually stopped. The temporary disruption during adaptation is illustrated in the figure. The adaptation used the policy from the “Specification-based Rule Framework” section.

Once the rules are enforced, monitoring resumes using the failover aggregator. In the second experiment (Figure 13b), a pure ECA based system was used and the adaptation framework enforced rules as soon as they were triggered. As illustrated in the figure, random enforcement of rules failed to connect the nodes to the failover aggregator and so the visualizer node did not receive data.

Before performing the second experiment, the original aggregator was restarted and the failover aggregator was disconnected. This caused no disruption in data reception since at least one aggregator was active during the switch. Thus, we see that with an ECPAP based approach, even though there is overhead in workflow generation, we guarantee recovery, whereas with a pure ECA based system (random enforcement order), complete disruption could happen.

Lessons Learned

Policy Design: Designing adaptation policies is an onerous task and requires significant knowledge of the

system configuration and functioning. Policy designers should foresee various changes that may affect IT systems and specify rules. This demands significant system knowledge and expertise from the administrator.

Defining Correctness: In this paper, we introduced the notion of enforcement semantics and identified three different semantics for rule enforcement. In order to prescribe a specific semantics we need to define correctness for concurrent rule enforcement. This requires empirical validation to determine if application of a specific semantics provides appropriate guarantees.

Optimization using Models: As discussed in the previous section, the workflow generation algorithms have high complexity. In a cluster of nodes, a single change may cause similar events from multiple nodes to be generated triggering multiple instances of the same rule. The workflow generation complexity can be greatly reduced if it can be inferred that the triggered rules are instances of a smaller set of rules by using configuration models. Similarly, using information models, such as Common Information Model (CIM) for representing system states enables faster evaluation of specifications. These models represent state information of entities and provide a central service that can be queried. Therefore, extending the adaptation framework with models enables optimization and we plan to explore that as extensions to our work.

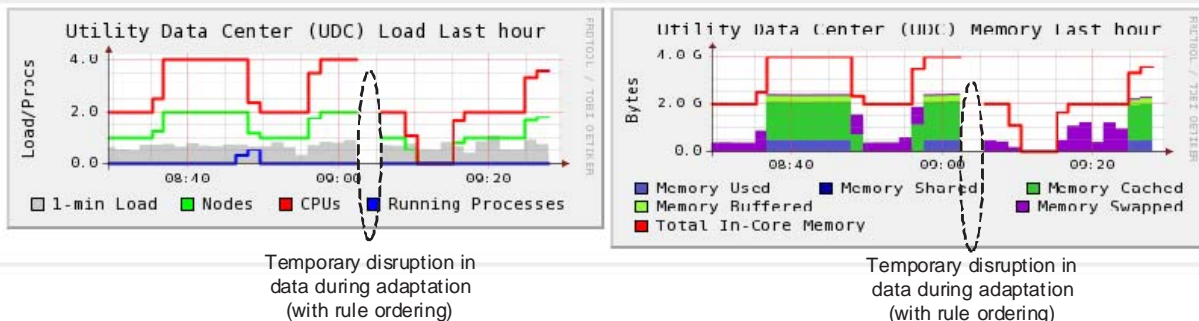


Figure 13a: Disruption in monitored data during adaptation (as perceived by the visualizer node) – ECPAP based system with Maximum Rule Enforcement (with reasoning).

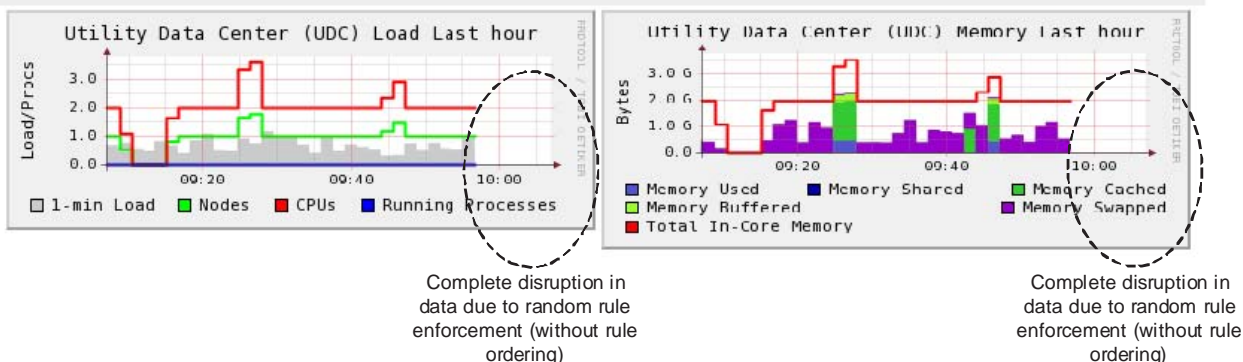


Figure 13b: Pure ECA based system (Random Rule Enforcement).

Event Correlation Models: Finally, the epoch model that we have used approximates a system change in our prototype and in order to develop a fully functioning adaptation system an event correlator is necessary. The correlator would require system information such as configuration, event delivery latency and so on, which can be represented as system models.

Related Work

Automated change management has gained attention in the past few years as a necessary technology to address the adaptation needs of rapidly growing enterprise and grid computing markets. Several research projects are focusing on reducing the administrator efforts in managing different aspects of large distributed systems through policy and model-based approaches.

The CHAMPS project [18] aims to reduce the complexity of IT change and configuration management in distributed environments through planning and scheduling approaches. The project uses model-based approaches to build task graphs to adapt to system changes. Our adaptation framework uses policies and therefore differs significantly from the CHAMPS project. Policy-based management has been used for network switch management [5], managing content distribution networks [6] and distributed systems [7].

The focus of our work is on managing complex IT systems. As we have described, the complexity of these systems causes simultaneous activation of multiple policy rules, which have to be enforced in proper order. None of the projects on policy-based management seem to address this problem, to the best of our knowledge, as we have addressed.

There have been several research efforts in designing policy languages [1, 15], detecting and resolving policy conflicts [2, 9, 11, 22], and various other analyses [8, 20]. To the best of our knowledge, no research in this area has addressed the problem of ordering management rules and providing enforcement guarantees as we have addressed in this paper.

Dunlop, et al. [22] use temporal characteristics of policies to dynamically reason about policy consistency. Their approach detects a large class of conflicts that cannot be detected statically. The focus of their work is on conflict analysis and not on ordering rule enforcement as we have presented in this paper. Our previous work [11] proposes an ECA-P framework to detect and resolve dynamic conflicts that occur due to side-effects of actions. The focus of the work was on conflict analysis and not on enforcement order determination.

Sloman, et al. [7, 8, 9] have developed the Ponder policy specification language and defined techniques for conflict analysis and role-based management. To the best of our knowledge, their work does not address the problem of ordering concurrently triggered rules.

Several research projects in autonomic computing reason about action ordering [12, 13]. These projects are based on AI planning techniques where users specify

high-level goals and the planning system determines the ordered set of actions to be executed to reach the desired goal state. The main difference between these projects and our work is that in goal-based approaches the final system state that needs to be reached is known and the system has to determine the actions to be executed to reach that state. In the problem that we have addressed, the final system state is unknown. When an event occurs, a set of rules get triggered and we need to reason about the execution order of the rule actions based on some enforcement semantics.

Finally, our application of policy-based techniques for adapting monitoring systems is a novel research effort. System monitoring is a well-researched field and several monitoring systems such as HP Openview [16], IRISLOG [25], Ganglia [14] and MonALISA [17] are currently being used. These systems focus mainly on data collection, delivery, scalability and fault tolerance. None of these systems support a generic framework that adapts to a spectrum of changes such as application migration, service plug-ins and so on, which is required for automated change management. Our framework uses the configurability features of these systems to adapt to changes based on administrator-specified policies.

Conclusion

IT systems are dynamic and subject to various changes. Management of such changes needs to be automated to reduce administration cost. Policy-based adaptation is a suitable approach where management actions are specified by an administrator, as Event-Condition-Action rules, for different changes in the system. The interdependence of components in modern IT systems causes several change events to be generated when a single change occurs triggering multiple rules. Since the order of enforcement of rules determines the system behavior, adaptation systems should reason about the enforcement order of the rules before initiating corrective actions. We found the ECA rule framework to be poorly suited for reasoning since it does not contain specifications of rule actions.

In this work, we introduced a new rule framework, called Event-Condition-Precondition-Action-Postcondition (ECPAP) that contains action specification for designing adaptation policies for IT systems. When multiple rules are simultaneously triggered on a change, the adaptation system uses the specifications to analyze dependencies between rule actions and generate a Petri net workflow that is executed by an execution engine. We introduce a new notion called enforcement semantics that provides guarantees about rule enforcement. We have used this framework for adapting monitoring systems to changes and presented its performance and advantages in this paper.

Acknowledgements

We would like to thank Martin Arlitt, Keith Farkas, and our shepherd John "Rowan" Littell, for

their comments, which have helped improve the content and presentation of the paper. We also thank Rob Kolstad for his extra typesetting efforts that have helped to significantly improve the look and feel of the paper.

Author Biographies

Chetan Shankar is a doctoral candidate at the University of Illinois at Urbana-Champaign (UIUC). He is one of the main contributors to the Active Spaces pervasive computing project at UIUC and has published several papers on pervasive computing and policy-based management. He is broadly interested in services and systems management, programming and management frameworks for dynamic systems and pervasive computing.

Vanish Talwar is a researcher in the Enterprise Systems and Software Lab at Hewlett-Packard Laboratories. His technical interests include distributed systems, operating systems, and computer networks, with a focus on management technologies. He received his M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana Champaign (UIUC) in 2001 and 2006 respectively. He is the recipient of the David J. Kuck Best Masters Thesis award in the Dept. of Computer Science, UIUC, and is an elected member of Phi Kappa Phi and Sigma Xi.

Subu Iyer is a Systems Software Engineer and researcher at HP Labs, Palo Alto. He joined DEC Network Systems Lab in 1997 where he worked on collecting and analyzing performance data from a large cluster of machines on DEC's Palo Alto Research Gateway. Over the years, Subu has worked on projects in the areas of distributed computing, performance monitoring and telepresence. His current work is on scalable adaptive performance monitoring.

Yuan Chen is a post-doctoral researcher in Enterprise Systems and Software Laboratory at HP Labs. He received a B.S degree from University of Science and Technology of China in 1994, and M.S. and Ph.D. degrees from the Georgia Institute of Technology in 2001 and 2005, respectively, all in Computer Science. His current research focuses on performance and systems management in complex and large-scale enterprise computing systems.

Dejan Milojicic is a senior researcher and a project manager at HP Labs. He has worked in the area of operating systems and distributed systems for more than 20 years. He has been the program chair of the IEEE Agent Systems and Applications Symposium (ASA/MA'99) and of the first USENIX Workshop on Industrial Experiences with System Software (WIESS'2000). Dr. Milojicic published in many journals and at various events. He is currently on the editorial board of IEEE Distributed Systems Online. He has been engaged in various standardization bodies, such as OMG and Global Grid Forum. He is a member of the ACM, IEEE, and USENIX. He received his B.Sc. and M.Sc. from University of Belgrade and his

Ph.D. from University of Kaiserslautern. Prior to HP Labs, Dejan worked at Institute "Mihajlo Pupin," Belgrade and at OSF Research Institute, Cambridge, MA.

Roy Campbell is the Sohaib and Sara Abbasi Professor of Computer Science at the Siebel Center for Computer Science at the University of Illinois, Urbana-Champaign. He has supervised the completion of forty Ph.D. dissertations and the author of over two hundred and forty four research papers on security, programming languages, software engineering, operating systems, distributed systems, and networking. His research includes the Gaia project on Active Spaces, the security of the power grid, and mobile computer operating systems. Professor Campbell is Director of the University Of Illinois Center Of Academic Excellence in Information Assurance Education, a member of the Information Trust Institute and, with Guy Garnett, directs the Cultural Computing Program. He is a member of the ACM and an IEEE Fellow.

Bibliography

- [1] Lobo, J., et al., "A Policy Description Language," *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pp. 291-298, July, 1999.
- [2] Chomicki, J., J. Lobo, and S. Naqvi, "Conflict Resolution Using Logic Programming," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, Num. 1, pp. 244-249, January/February, 2003.
- [3] Reisig, W., *Petri Nets: An Introduction*, Springer-Verlag, New York, 1985.
- [4] Roussev, B. N., "Self-checking Implementation of Boolean Interpreted Petri Nets," *Proceedings of IEEE Symposium on Emerging Technologies and Factory Automation*, 1994.
- [5] Bhatia, R., et al., "Policy Evaluation for Network Management," *Proceedings of 19th Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 2000)*, pp. 1107-1116, March, 2000.
- [6] Amiri, K., et al., "Policy Based Management of Content Distribution Networks," *IEEE Network Magazine*, 2002.
- [7] Sloman, M., "Policy Driven Management For Distributed Systems," *Plenum Press Journal of Network and Systems Management*, Vol 2, Num. 4, Dec., 1994, pp. 333-360.
- [8] Lupu, E. C., *A Role-Based Framework for Distributed Systems Management*, Ph.D. Thesis, Imperial College, London, 1998.
- [9] Lupu, E. C., et al., "Conflicts in Policy-Based Distributed Systems Management," *IEEE Transactions on Software Engineering*, Vol. 25, pp. 852-869, Nov., 1999.
- [10] Shankar, C., et al., "A Policy-based Management Framework for Pervasive Systems using Axiomatized Rule Actions," *Proceedings of Fourth*

- IEEE International Symposium on Network Computing and Applications (IEEE NCA05)*, MA, 2005.
- [11] Shankar, C., et al., "An ECA-P Policy-based Framework for Managing Ubiquitous Computing Environments," *Proceedings of The Third Annual International Conference on Mobile and Ubiquitous Systems: Networks and Services (MobiQitous 2005)*, San Diego, July, 2005.
 - [12] Andrzejak, A., et al., "FeedbackFlow – An Adaptive Workflow Generator for System Management," *Proceedings of The Second IEEE International Conference on Autonomic Computing (ICAC-05)*, June, 2005.
 - [13] Srivastava, B., et al., "The Case for Automated Planning in Autonomic Computing," *Proceedings of The 2nd IEEE International Conference on Autonomic Computing (ICAC-05)*, June, 2005.
 - [14] Massie, M., B. Chun, and D. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, Vol. 30, Issue 7, July 2004.
 - [15] Damianou, N., et al., "The Ponder Specification Language," *Proceedings of Workshop on Policies for Distributed Systems and Networks (Policy2001)*, HP Labs Bristol, pp. 29-31, Jan., 2001.
 - [16] *HP OpenView Management Solutions*, <http://www.managementsoftware.hp.com/>.
 - [17] Newman, H. B., et al., "MonALISA: A Distributed Monitoring Service Architecture," *Proceedings of 2003 Conference for Computing in High Energy and Nuclear Physics (CHEP03)*, La Jolla, California, March, 2003.
 - [18] Brown, A., et al., "A Model of Configuration Complexity and its Application to a Change Management System," *Proceedings of the 9th International IFIP/IEEE Symposium on Integrated Management (IM 2005)*, May, 2005.
 - [19] *XSB Logic Programming and Deductive Database system for UNIX and Windows*, <http://xsb.sourceforge.net/>.
 - [20] Verma, D., "Simplifying Network Administration using Policy based Management," *IEEE Network Magazine*, 2002.
 - [21] Baralis, E. and J. Widom, "Better Static Rule Analysis for Active Database Systems," *ACM Transactions on Database Systems*, Vol. 25, Num. 3, pp. 269-332, September, 2000.
 - [22] Dunlop, N., et al., "Dynamic Conflict Detection in Policy-Based Management Systems," *Proceedings of Enterprise Distributed Object Computing Conference (EDOC '02)*, September, 2002.
 - [23] Hoare, C. A. R., "An axiomatic Basis for Computer Programming," *Communications of the ACM*, Vol. 12, Num. 10, 1969.
 - [24] Ranganathan, A., et al., "Pervasive Autonomic Computing Based on Planning," *Proceedings of IEEE International Conference on Autonomic Computing (ICAC-04)*, May, 2004.
 - [25] Nath, S., et al., "Tolerating Correlated Failures in Wide-Area Monitoring Services," Intel Research TR, May, 2004.
 - [26] Klinger, S., et al., "A Coding Approach to Event Correlation," *Proceedings of the 4th International IFIP/IEEE Symposium on Integrated Management (IM 1997)*, 1997.
 - [27] OpenView Event Correlation Service, <http://www.managementsoftware.hp.com/products/ecs/>.

Appendix I

Definition 4: Formally, the BIPN of a set of actions $A = \{a_1, \dots, a_n\}$ is a 1-safe marked Petri net [4] represented as a triple $B = (P, T, F)$ where

$P = \{ \text{place}(a) \mid \forall a \in A \} \cup \{ \text{Start} \}$, where $\text{place}(a)$ is the place representation of action a .

$T = \{ t_{K, \text{pre}(a)} \mid \forall x \in K, t_{K, \text{pre}(a)} \in x^\bullet \wedge \text{place}(a) \in t_{K, \text{pre}(a)}^\bullet \}$, where K is a set of places and for $x \in P \cup T$, $x^\bullet = \{y \mid yFx\}$ is called the *input set* of x and $x^\bullet = \{y \mid xFy\}$ is called the *output set* of x and the flow relation, $F \subseteq (P \times T) \cup (T \times P)$ such that $\text{dom}(F) \cup \text{codom}(F) = P \cup T$. $\text{pre}(a)$ represents the pre-condition of action a .

The Petri net generated from Algorithms 1-3 for actoin set A is represented as $B = (P, T, F)$ where

$P = \{ \text{place}(a) \mid \forall a \in A \} \cup \{ \text{Start} \}$

$T = \{ t_{i,j} \mid (i = \{ \text{Start} \}, j = \text{true}) \wedge$
 $(i = \{ \text{place}(a) \}, j = \text{pre}(b) \mid \forall a, b \in A, \text{post}(a) \models \text{pre}(b)) \wedge (\text{pre}(b) \not\models \text{true}) \wedge$
 $(i = s, j = \text{pre}(b) \mid \forall b \in A, (\forall s \in 2^{P - \{ \text{Start} \}}, \bigwedge_{k \in s} \text{post}(\text{action}(k))) \models \text{pre}(b)) \},$
 $\text{action}(k)$ represents the action in set A assigned to place k .

$F = \{ (x, y) \mid \forall t_{i,j} \in T, \forall x \in i, y = t_{i,j} \} \cup$
 $\{ (x, y) \mid \forall t_{i,j} \in T, (x = t_{i,j} \wedge y = \text{place}(k), \forall k \in A \mid j = \text{pre}(k)) \}$

The three conjuncts in the definition of T correspond to the transitions resulting from Algorithms 1-3. The transitions are labeled $t_{i,j}$ where $i = \bullet t_{i,j}$ and j is the assigned Boolean function. The flow relation, F , represents the various edges of the Petri net.

Theorem 1: For a set of actions $A = \{a_1, \dots, a_n\}$, the Petri net generated by Algorithms 1-3 enables maximum number of actions starting from the current system state I .

Proof. To prove the above theorem, it is sufficient to prove that for every action $a \in A$, if $I \Rightarrow_k a$, then there is a reachable path [3] in the Petri net from the *Start* place to $\text{place}(a)$, where $I \Rightarrow_k a$ means that starting from the current system state I , successful execution of k actions of A enables a . $X \rightarrow a_1$ implies execution of all actions of set X enables a_1 . We prove this by structural induction on the Petri net.

Basis: $I \Rightarrow_0 a$

$\text{pre}(a)$ is satisfied by current system state and so a is trivially-enabled by Algorithm 1. Therefore, $t_{\{ \text{Start} \}, \text{true}} \in T$ and $\{ (\text{Start}, t_{\{ \text{Start} \}, \text{true}}), (t_{\{ \text{Start} \}, \text{true}}, \text{place}(a)) \} \subseteq F$. Therefore, there is a reachable path from S to $\text{place}(a)$ through the transition labeled $t_{\{ \text{Start} \}, \text{true}}$.

Hypothesis: Assume if $I \Rightarrow_k a$ there is a reachable path from *Start* to $\text{place}(a)$. We need to prove that if $I \Rightarrow_{k+1} a_1$ there exists a reachable path from *Start* to $\text{place}(a_1)$.

Since $I \Rightarrow_k a$ from our inductive hypothesis, there is a set of actions $A' \subset A$ such that $\forall x \in A', I \Rightarrow_{\leq k} x$ and $A' \rightarrow a_1$. Therefore, there is a reachable path from *Start* to $\text{place}(x)$ for all $x \in A'$. There are two cases to consider.

Case 1: $A' = \{a\}$ Since a is found to enable a_1 from enablement analysis in Algorithm 2, $t_{\{ \text{place}(a) \}, \text{pre}(a_1)} \in T$ and $\{ (\text{place}(a), t_{\{ \text{place}(a) \}, \text{pre}(a_1)}), (t_{\{ \text{place}(a) \}, \text{pre}(a_1)}, \text{place}(a_1)) \} \subseteq F$. Therefore, there is a reachable path from $\text{place}(a)$ to $\text{place}(a_1)$ and since by hypothesis there exists a reachable path from *Start* to $\text{place}(a)$, by transitivity, there is a reachable path from *Start* to $\text{place}(a_1)$.

Case 2: $\text{Cardinality}(A') > 1$ Actions in A' are found to enable a_1 from partial-enablement analysis in Algorithm 3. Therefore, $t_{\{ \text{place}(x) \mid \forall x \in A', \text{pre}(a_1) \}} \in T$ and

$\{ (\text{place}(a) \mid \forall a \in A', t_{\{ \text{place}(x) \mid \forall x \in A', \text{pre}(a_1) \}}), (t_{\{ \text{place}(x) \mid \forall x \in A', \text{pre}(a_1) \}}, \text{place}(a_1)) \} \subseteq F$.

Therefore, there is a reachable path from $\text{place}(x)$, $\forall x \in A'$ to $\text{place}(a_1)$ through the transition $t_{\{ \text{place}(x) \mid \forall x \in A', \text{pre}(a_1) \}}$. Since there is a reachable path from *Start* to $\text{place}(x)$, $\forall x \in A'$ from our hypothesis, by transitivity, there is a reachable path from *Start* to $\text{place}(a_1)$.

Experience Implementing an IP Address Closure

Ning Wu and Alva Couch – Computer Science Department, Tufts University

ABSTRACT

Most autonomic systems require large amounts of human labor and configuration before they become autonomous. We study the management problem for autonomic systems, and consider the actions needed before a system becomes self-managing, as well as the tasks a system administrator must still perform to keep so-called “self-managing systems” operating properly. To understand the problem, we implemented a prototype self-managing “IP address closure” that implements integrated DNS and DHCP. We conclude that the system administrator is far from obsolete, but that the administrator of the future will have a different skill set than those of the present, focused around effective interaction with closures rather than management of individual machines.

Introduction

Imagine that you are asked to set up a new DHCP/DNS infrastructure. You proceed to collect a pocket-full of “Ethernet keys” that look like USB keys, but each contains a micro-controller and an Ethernet interface, where power is drawn from the Ethernet plug. You proceed to plug one of these keys into a test network and give it a specification of your network architecture in the form of an operating policy. Then you plug in the other keys to the same network, and each copies the policy from the first key. Finally, you unplug some of the keys and plug one or more keys into each Ethernet subnet and *voila*, you have a self-managing IP address infrastructure that is self-healing, and in which telling any key about a policy change causes that change to propagate to the whole infrastructure of keys. If a key dies, you replace it with another key that has – for awhile – been plugged into any subnet containing a working key. No backups are necessary; the infrastructure is completely self-managing and self-healing.

Are we dreaming? Not really, as this paper will show. It is possible to implement such devices and infrastructure. But there is a larger question that we have not addressed: what happens when something goes wrong? The subtlety in managing such an infrastructure lies in the interface between the keys and the human administrator. When things go wrong, a human is still required to intervene and repair problems.

Closures

Our implementation of the above-mentioned behavior is based upon the theory of closures. A *closure* [3] is a self-managing component of an infrastructure that protects one part of IT infrastructure while making its needs known to other closures [20]. The closure model of a managed system expresses the system as a composition of communicating components. In previous experiments on closures [20], it has been demonstrated that any high-level closure needs

support from other low-level closures. For example, consider a web service. The service itself can be encapsulated within a closure, but does not handle IP address assignment and DNS [15, 16]. These functions must be handled via one or more lower-level closures in a self-managing infrastructure.

In this paper, we describe experience and lessons learned in building and testing an “IP address closure.” This closure is a self-managing “fabric” of distributed nodes that handles DNS and DHCP for an infrastructure. The IP address closure handles address assignment based upon three inputs: requests for addresses, a policy on address assignment, and architectural information about routing and gateways within the network. The IP address closure sits between the web service closure and a routing closure (which may be implemented by a human being), accepting inputs from both (Figure 1).

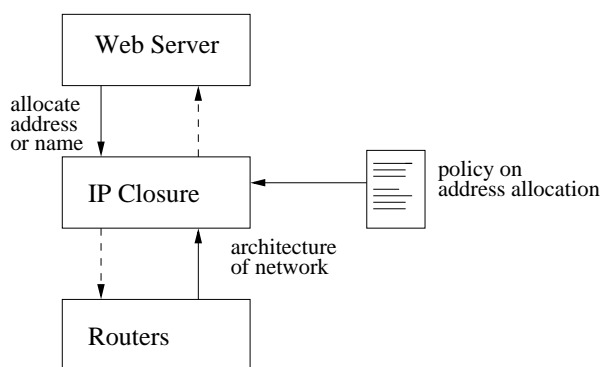


Figure 1: Interaction between closures.

A New Management Style

Managing the IP address closure is very different than managing the Web service closure. The web service closure is managed via “commands” that change state of a single server or web farm. By contrast, the IP address closure fabric is composed of small, movable

“black boxes” that can serve as DHCP and/or DNS servers. These are configured by a process of *seeding*. Each box is initialized by physically plugging it into the same subnet as an already seeded box, by moving it physically. The new box discovers the existing seeded box, clones its configuration, and receives an idea of network topology, policy, and locations of peers from the existing box. After this, it is moved to its final physical location, after which it can serve to seed other boxes.

Simple Hardware Components

An element of the IP address closure is an extremely simple device with an Ethernet connection and some form of persistent storage. It is conceivable that a closure node could be implemented in hardware using only non-moving parts such as flash and regular memory (no hard disk would be required), thus leading to extremely low hardware cost for the self-managing fabric. It is even possible to power a node from the Ethernet connection, so that it can be a completely self-contained device similar to a USB key (an “Ethernet key”). We foresee a time in which IP management could literally be accomplished by a pocket full of keyring-sized devices, carried from room to room as needed. A similar approach, using the same environmental discovery and arbitration algorithms, could be used to create closures for other tasks such as distributed monitoring, intrusion detection, troubleshooting, web caching, file system caching, and secure remote access.

Backup and Recovery

This is a dramatic difference in how one provides failover and recovery in the IP address closure fabric, compared to managing current DNS and DHCP servers. To establish redundancy on a subnet, one simply plugs another box into the subnet, and the new box makes itself a clone of the boxes it discovers, to become a backup server. If one unplugs a box, any backup servers automatically start serving requests. If a box fails, one simply unplugs it and plugs in another. The boxes serve as their own backups; any box is interchangeable with any other in case of failures. Each box discovers where it is operating, and how many neighbors it has, before deciding to provide services or serve as a backup server. Thus backups are as easy as keeping a spare box that one plugs into a subnet periodically in order to keep the backup node up to date, and recovery is a matter of plugging the backup node back into the network so that its changes can be propagated to other nodes.

Low-level Last

It may seem to the reader that we have gone about the problem of building closures “backwards”; previous authors have studied “high-level” closures that – to operate properly – require low-level closures that we tend to implement *after* the closures that utilize them. The reason for this backward implementation order is that many of the challenges in building a

closure come into play at the lowest levels, where the interface between the system administrator and the closure is most complex. At the lowest level, closures are limited by the fact that software cannot accomplish physical changes in hardware or network configuration. When configuring a web server [20], this is not much of a concern, but at the IP level, it is a central issue.

Related Systems

In large-scale systems, manual procedures for maintaining static and dynamic IP address assignment are both tedious and error-prone. IP management tools have been developed to help administrators manage the IP space in an enterprise; CISCO Network Registrar [2], INS IPControl [6], and Lucent VitalQIP [13] are examples of current products. Common features of IP management software include integrated DHCP and DNS service, centralized policy management, and failover mechanisms for high availability. These products require crafting of detailed IP assignment policies, as well as manual configuration of all nodes included in the service. Melcher and Mitchell [14] mention the need for an autonomic solution for DHCP, DNS, LDAP, and other services. It is also highly desirable to minimize the amount of human input necessary to configure the system, avoiding the “incidental complexity” of making policy decisions that have no externally observable behavioral consequences [3].

Goals

Our goals in creating the IP address closure were to help administrators by:

- Encapsulating a reusable design of the IP assignment plan in a policy.
- Reducing incidental complexity by automating unimportant decision-making.
- Automating the process of implementing changes in policy.
- Providing autonomic features such as self-configuration, self-backup, and self-healing.
- Simplifying day-to-day management of the IP address (DHCP/DNS) infrastructure.

The IP address closure can be seen as an effort to implement autonomic features [4, 5, 10] in the IP layer.

Paper Organization

In this paper, we will use the IP address closure as an example of the potential impact of autonomic systems upon system administrators, and show that system administrators can benefit from it and similar systems. Far from threatening the jobs of system administrators, the IP address closure is instead a “partner” that requires ongoing management, in return for offloading some common management tasks.

This paper is organized as follows. We begin by describing the overall design and function of the IP address closure. We then discuss the design and implementation details for the IP address closure and critique our prototype. We subsequently discuss the relationship

between autonomic systems and system administration and then discuss the issue of exception handling. Finally, we conclude this paper and discuss future work.

Closure Design

The design of our IP address closure is so unlike that of any prior work that a detailed discussion of its theory of operation is necessary. In this section, we give a detailed theory of operation intended to convince the reader that the closure will work as described. The closure's theory of operation is somewhat subtle, and this section can be skipped without loss of continuity if the reader is not interested in implementation details.

Peer-Peer Architecture

Unlike prior closures, which resided primarily on one machine, the IP address closure resides within a peer-peer "fabric" of distributed "black boxes" that manage the state of the IP layer for an enterprise. These "Peered IP" management nodes, or "PIPs," manage themselves based upon a high-level policy and environmental factors that the PIPs discover through direct probing of their environments. PIPs can be implemented as small and cheap "Ethernet appliances" that support each other and implement both self-healing and self-replication features.

A peer-to-peer solution is more robust and easier to use; there is no need to manage a centralized database. The distributed nodes have a better view of the environment than a central probe; they can see through firewalls and other protections, and can acquire environmental information [12] that is more accurate than relying upon human input. If we tell one peer about a new policy, it distributes the policy to all of its known peers, which continue relaying the policy until it is present on all nodes. However, control of information distribution is more difficult than in the centralized case. For example, at any particular point in time, there can be conflicts between the policy information in replicas. A policy change must be broadcast to all the PIPs.

It would have been nice if we could have utilized an existing peer-peer scheme for implementing our closure. The drawback of utilizing existing peer-peer schemes is that their own bootstrap protocols require prior existence of a stable IP layer. Also, their complexity and goal of distributing large amounts of information is much more ambitious than we need. We utilize a simple pull-only gossiping protocol to communicate relatively brief policy information among PIPs, after a (rather complex) bootstrap and environment discovery protocol that is necessary because there is no IP layer before the closure becomes functional.

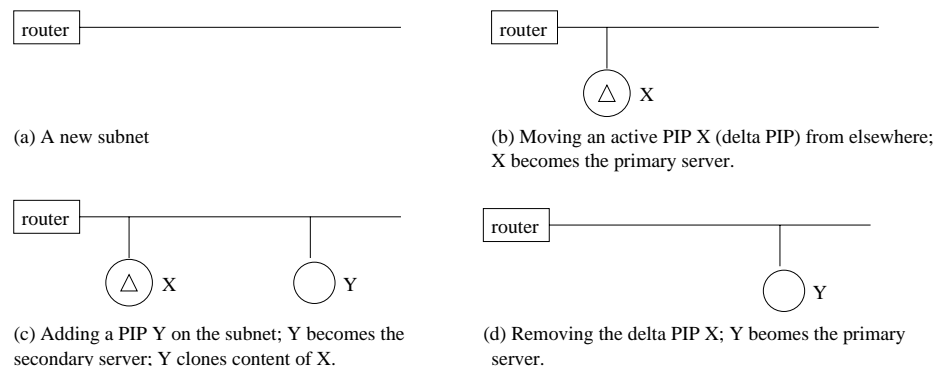


Figure 2: Bootstrapping a new subnet from a delta PIP.

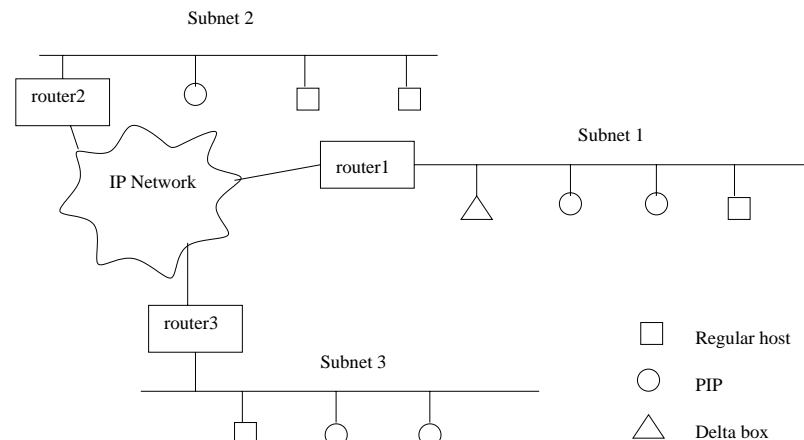


Figure 3: An example of how PIPs can be deployed.

In practice, using a complete peer-to-peer environment poses problems in network design. If a network policy chooses to maintain separate subnets, it may still allow the DHCP servers to talk to a central server, and vice versa. If administrators choose to use a peer-to-peer architecture, the complexity of firewall rules will be increased. The deployment of web services [23] faces similar issues. If lower-level network closures exist, the requirement of configuring firewalls can be delegated to it; if not, administrators must manually configure the firewalls.

Bootstrapping the Closure

The most innovative feature of our IP address closure is how it bootstraps itself into a functional state on a previously unmanaged network. Unlike other closures and autonomic computing solutions, our closure must be able to cope with a network where the IP layer is not yet functional. This leads to a rather unique process for bootstrapping, based upon policy seeding and environmental discovery. There are three types of hosts in an IP address closure: regular host, PIP, and delta box (Figure 3). Regular hosts are the clients of the DHCP service provided by PIPs. PIPs are the management nodes that provide DHCP and DNS services. A delta box is a special type of PIP that potentially contains different information from other PIPs on the same subnet; otherwise, it is the same as a generic PIP. A delta box can be used to deliver information to another subnet by connecting it physically to that subnet. This feature is very useful for distributing policies to networks that are physically segregated from the rest of the infrastructure, e.g., by firewalls.

The bootstrapping process for the IP address closure is different from that for a normal system. A PIP, referred to as “delta PIP,” can be moved *physically* through the network to propagate information about changes. Bootstrapping of the closure is done by a combination of logical specifications and physical moves of devices. To bootstrap the closure, one starts with a policy file and loads it into a single delta PIP. Connecting this box to each segment of the network discovers other peers on that segment and communicates the policy to them. The delta box also records and distributes the addresses of other peers. At the end of the seeding process, every node knows about every peer, and the delta box is no longer needed. The bootstrapping process is depicted in Figure 2.

Figure 3 shows an example of the deployment of PIPs. The IP network could contain many firewalls between routers, and the subnets can even be disconnected with no ability to communicate, provided that physical moves are utilized to propagate policy changes. Assume subnet one is separated from subnet two and subnet three. Subnet one has three PIPs deployed (two PIPs and one newly arrived delta box). Subnet two has only one PIP, so there is no failover backup on subnet two. Subnet three has two PIPs that will form a failover pair.

The one command that every PIP must understand is ‘dump.’ Once a PIP receives a dump request, it adds the requesting PIP into the known PIP list and dumps its knowledge base to the requester. It is the job of the requester to analyze the content and change its own configuration accordingly. Each PIP periodically probes other selected PIPs in the known PIP list. The PIPs probed are chosen according to the structure of the P2P network; one box per subnet is arbitrarily chosen. If a PIP cannot be contacted in a specified period of time, it is removed from the neighbor/known PIP list.

The freshness of information is controlled by versions consisting of time stamps. Each piece of information is stamped with the clock time of the change observer (which might not be a local peer).

Policy Change Planning

Each PIP records its own decisions in a low-level operational policy file. When another PIP appears on the same subnet, it might take over some tasks because of performance or other reasons, and mark the operational low-level policy accordingly. The functional states of PIPs in a particular subnet are managed by a “boss” PIP, whose identity is determined by a booting race condition. Only the “boss” of a subnet can change the low-level behavioral attributes related to that subnet. The “boss” effectively acts as a coordinator to prevent write conflicts.

We must assure that policy changes propagate to every PIP, and that there is a global convergence among the PIPs to one coherent overall policy. This means that all the active PIPs in the IP address closure either accept or reject a high-level policy together. Before a high-level policy is used, a policy proposal is published to the closure. Then the PIPs decide whether the proposal is feasible. Our research does not focus on how to quickly reach consensus in a distributed environment; we choose a simple two-phase protocol and leave the problem of optimizing this protocol to future work. The good news is that because the IP address closure is operating in a controlled environment, the complexity of the consensus problem is significantly reduced.

A high-level policy proposal is accepted only when all active PIPs vote ‘yes,’ which indicates that all the preconditions in this policy that are related to a particular PIP are satisfied. It is possible that a change is rejected and a PIP votes ‘no.’ This happens when the physical constraints of the policy are violated. For example, one fixed mapping in the policy file might be impossible because the host is physically on a different subnet than the policy expects.

Self-healing and Failover

While the self-healing features are implemented by redundancy, there are some special considerations for self-healing of DHCP and DNS servers. In particular, any failover should allow for:

1. Seamless management of leases granted by the failed peer.
2. Address spoofing of failed DNS servers during failover.

In our prototype, we handled the first condition but not the second; it is reserved for future work.

In order to provide failover DHCP service from one server to another server, IP leases must be cached somewhere else so they can be managed on a new server. One way to do this is to store the leases in a P2P infrastructure (for example, openDHT [18]). In this way, every IP assignment is recorded in the network, and transitioning from one server to another is easy, because the information is not stored in each individual server alone; replicas are stored in the P2P network. We chose to use the existing DHCP Failover protocol [17], implemented by ISC DHCP. This failover protocol meets most of our goals but has a constraint that it only supports failover in pairs. This constraint limits the number of backup servers to one at any given time. Redundant backup servers are on standby, awaiting future need.

Failures could be in hardware, network, software, etc. The goal of redundancy is to keep the DHCP and DNS service running whenever possible. If a server starts functioning again after a failure, it should be able to recover from the failure; if it fails permanently, the service should still be provided if possible. In the current failover protocol, if a failover pair cannot communicate with each other, they split and share the IP pool for new IP assignment until the communication recovers, because a PIP does not know whether the failure is due to network failure or node failure. If the network partitions and both primary and secondary DHCP servers are assigning IP

addresses without splitting, there may be conflicts when a PIP rejoins the network after a long absence. Currently, we are satisfied with the solution of notifying system administrators when the failover mechanism is invoked. If human administrators determine that one of the servers has indeed failed, a backup server can be added to the subnet.

Bootstrapping a PIP

Each PIP acts as a primary DHCP server, secondary DHCP server, or backup DHCP server. A booting state diagram (Figure 4) shows the states of a PIP when it boots. The booted PIP can be in several states depending on the network environment. If it obtains an IP address from a DHCP server, it will enter the ‘cloning’ state, in which policies are dynamically kept synchronized with the current segment server. If it does not receive a DHCP response and discovers its own IP number from its environment and policy, it will assume that it is alone on the network segment and go into active service as a DHCP server. Else, if a PIP cannot determine its IP address by any means, the boot process fails.

During bootstrapping, a PIP must determine the segment into which it has been plugged. It first sends a DHCP request message on the segment, hoping a DHCP server will respond and assign it an IP address. If not, it probes the network and determines its location, and then assigns itself an IP address based upon that probe.

How does a PIP determine its own IP address if DHCP is not yet running and it is the first potential server? Ideally, we should be able to obtain this location information from lower-level closures – for example, through a broadcast-based protocol. Without such a luxury, we must probe for an IP address that we can

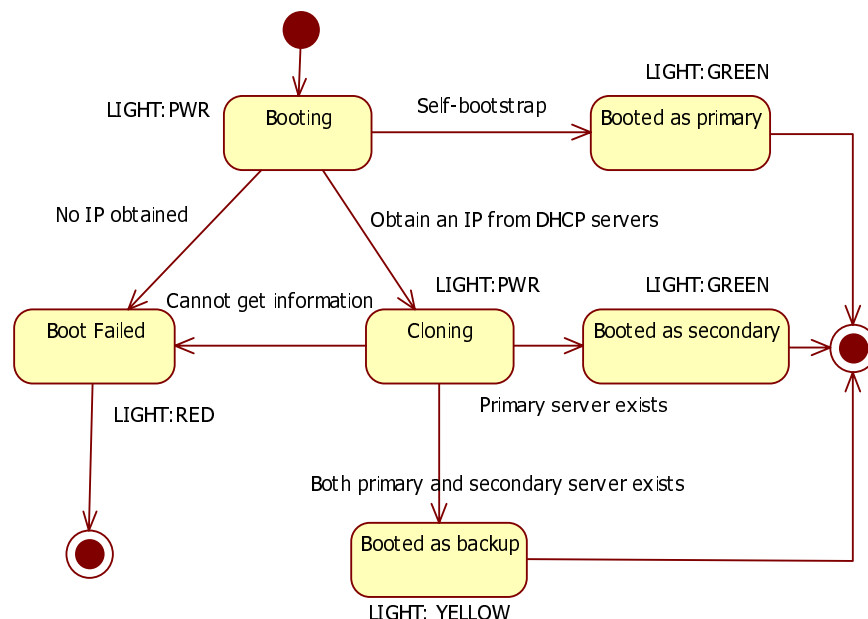


Figure 4: The booting state diagram.

use to exchange information with other nodes. We implemented the probe mode in our prototype. For example, we have the following definition for segment 192.168.3.0/24 in the seed file:

```
<seed>
  <segment>
    <network>192.168.3.0</network>
    <cidr>24</cidr>
    <router>192.168.3.1</router>
    <bootip>192.168.3.2</bootip>
  </segment> ...
</seed>
```

Each node actively probes to determine which segment in the list of possible segments is directly connected to it. The seed file contains a list of primary routers, with one unused IP address (called the bootstrap IP) for each segment. A PIP iterates through the segments and try to ARP the corresponding primary router. If it receives an ARP reply from the router

within a specified period of time, then it concludes that it is connected to the corresponding subnet.

In using the probe protocol, a race condition can occur when two PIPs are bootstrapping on the same segment simultaneously. Then both PIPs try to use the same IP address. To avoid the race condition, each node sends an ARP request to resolve the bootstrap IP address. We refer this kind of ARP request as a *claiming ARP*, because the goal of this ARP request is to claim that a node is going to use the bootstrap IP address. If this IP address is already used, the node will receive an ARP reply from the bootstrap IP address, indicating that this address is already in use by another host. Then the booting node will simply abort the bootstrapping process.

If, after a period of time, no other claiming ARP request for the bootstrap IP address is received, the PIP will assign itself that address (we will call this

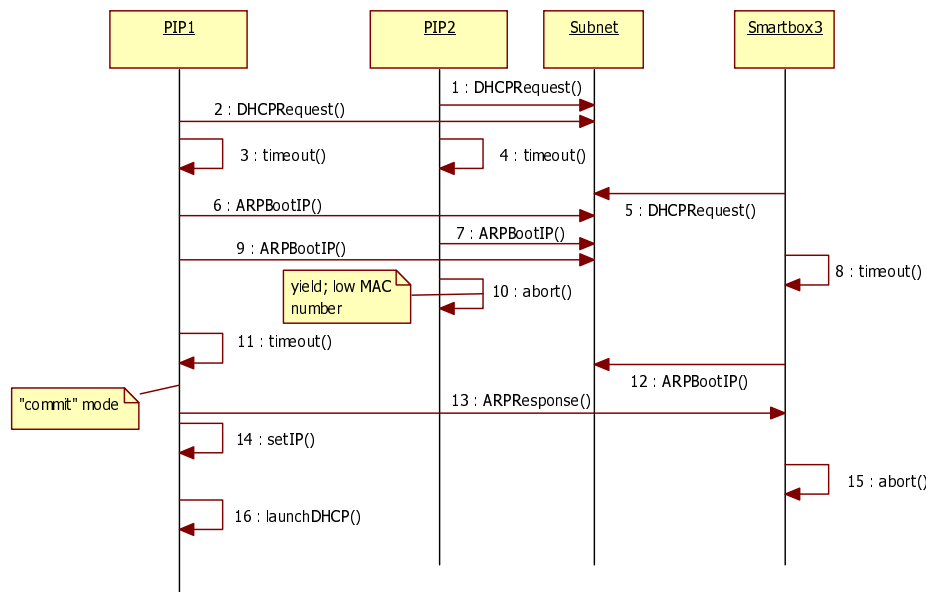


Figure 5: The bootstrapping sequence diagram. PIP1 wins in the bootstrap competition.

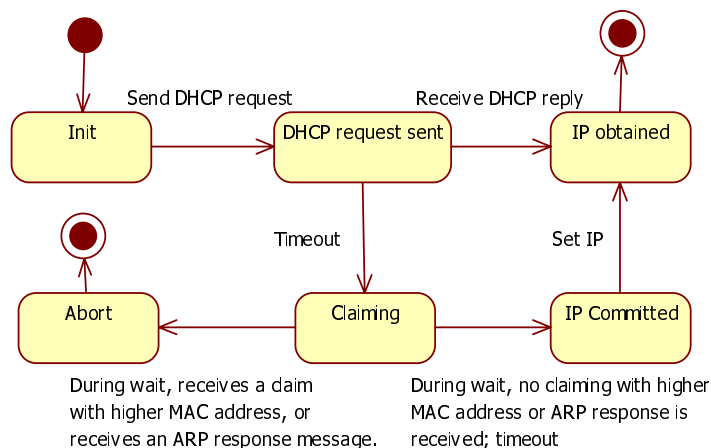


Figure 6: The bootstrapping state diagram.

state ‘committed’). If before this timeout (commit) event, more than one PIP is booted at the same time, each will receive the claiming ARP request at roughly the same time; the winner is determined based on MAC address. The PIP with a higher MAC address proceeds while PIPs with lower MAC addresses yield quietly. However, if one PIP has already committed its IP address, it will send a ARP reply claiming the IP/MAC mapping, as if it is already using that IP address. Any PIP, even though it may have a higher MAC address, will yield when it receives such an ARP reply, because that means the IP has been taken.

Before the IP address on the PIP is committed, the bootstrapping program is responsible for sending ARP responses so other nodes will yield. After the IP address is set, the ARP response will be generated by the OS. In this bootstrap protocol, the timeout period must be long enough to guarantee the ARP response is received if there is another host using the same IP address. Note that in this protocol, no incorrect ARP replies are sent to the network, so no ARP poisoning is caused by our protocol. Figure 5 shows a sequence diagram of three PIPs trying to boot at about the same time. A state diagram (Figure 6) shows the state transitions for this protocol.

Seed File Distribution

To minimize the work of the system administrator, we designed a mechanism to help with distribution of the seed file. We achieve this goal via a seed PIP, which is a delta PIP, indicating that it moves between subnets to gather information from PIPs in each one. The seed PIP first self-boots, then provides DHCP service. When a second PIP is plugged into the network, it gets an IP address via DHCP from the seed PIP. Then it configures itself as a failover for the seed PIP. In turn, the seed PIP can be removed without affecting the service, and moved to another subnet, where the process repeats.

Once the seed files are copied, a seed PIP is no different from other PIPs. The administrator can unplug any of the PIPs on the current net and use it as a seed PIP on a different subnet. We intend for the PIP to eventually have a light weight and a small size, so it can be carried easily around to seed other PIPs, e.g., behind firewalls.

The State Transition Problem

A system is rarely static. During its lifecycle, humans will request many changes in system behavior. System administrators need to be able to move the system from one operational state to another. This is called the *state transition problem*.

Traditionally, humans have been in charge of state transitions. The human administrator manipulates each device (in some way) into a new state. When systems become self-managing, however, it is possible for the systems themselves to take an active role in changing states. The ideal situation occurs when the system

being self-managed knows the best possible way to change state, so that it serves as a “co-pilot” or “partner” to the system administrator requesting the change.

In autonomic computing, several change planning systems have been developed – for example, CHAMP [9]. CHAMP is a task scheduling system that tries to optimize task scheduling based on cost and time constraints. CHAMP differs markedly from the IP address closure. It tries to solve the scheduling problem for changes so that downtime and disruption are minimized, and distributes the tasks for parallelism; and the calculation is centralized on a single master host. By contrast, the IP address closure does not compute change schedules centrally. Change schedules for the IP address closure can be computed locally, because the IP assignments for different subnets do not depend on one another.

Using the Closure

This section describes how one uses the closure. Low-level closures such as this one pose unique challenges for the system administrator. For the IP address closure to be functional, the system administrator must synthesize a description of its operating environment as well as its operating policies. There is an intimate relationship between contents of the closure configuration file and the routing architecture of the site. Thus the human administrator is in no sense obsolete; changes in the environment must be made known to the closure on a continuing basis.

Our IP address closure’s input is a policy file describing the desired relationships between IP numbers, network names, MAC addresses, and subnets. For example, it specifies which subnets are present and their numbering ranges. Some of this information would ideally be determined by a lower-level routing closure, e.g., the addresses of subnet gateways; here we (temporarily) encode that information into a seed file instead.

When using the IP address closure, the only thing a system administrator must specify is the intended behavior of the IP space; one is relieved from managing superfluous and “incidental complexity” with no behavioral impact [3]. For example, the tedious task of insuring agreement between DHCP servers on the location of routing gateways is managed by the closures and the human administrator need not participate.

The Policy File

In the IP address closure, there are two levels of policy. The first is a high-level policy that defines the overall behavior of the closure and reflects the IP scheme of the whole organization. This is determined by the system administrator. The second is a low-level policy that describes the behavior of the running system and how actual configuration files are generated. This is determined by the closure itself. For example,

the number of hosts allowed in a certain subnet is part of a high-level policy, whereas which host serves as primary and which serves as secondary failover server is part of a low-level policy. The high-level policy file contains the DHCP pools of available public IP addresses and private IP addresses, physical subnets, lease period, and some strategies about how the IP address closure is formed. These attributes define the behavior of the closure. The high-level policy specifies the goals of a bootstrap, while the low-level policy represents a steady operating state in which the bootstrap has been accomplished.

The high-level policy file reflects the IP scheme of the whole organization. Some part of the high-level policy may not be realized by a particular closure. Before a new version is released, it can be validated by several rules, including checks for consistency, IP overlapping, and syntax errors. After validation, a new policy will be broadcast to all the servers in the closure. The following code shows an example high-level policy file.

```
<policy ts="1136488200">
  <!-- static mapping from MAC to
                                IP address -->
  <include tag="mac-to-IP">
    fixed-ip.xml</include>
  <!-- static mapping from MAC to
                                host name -->
  <include tag="mac-to-name">
    fixed-name.xml
  </include>
  <!-- will be maintained by a
                                router closure -->

  <topology>
    <!-- defines subnets connected
                                by DHCP relay agents -->
    <relayed-subnet id="department1">
      <subnet>192.168.1.0</subnet>
      <subnet>192.168.5.0</subnet>
    </relayed-subnet>
  </topology>
  <pools>
    <pool access="private">
      <from>192.168.0.0</from>
      <to>192.168.254.0</to>
      <cidr>24</cidr>
      <max-lease-time>51000
    </max-lease-time>
    <subpool>
      <from>192.168.3.10</from>
      <to>192.168.3.254</to>
      <cidr>24</cidr>
      <max-lease-time>510000
    </max-lease-time>
    <include tag="restriction">
      res.xml</include>
    </subpool>
  </pool>
</pools>
  <!-- Special rules (exceptions)
                                to the previous rules -->
  <!-- Some rejected hosts -->
  <include tag="rejected-hosts">
    blacklist.xml</include>
  <!-- Some VIP hosts -->
  <include tag="VIP">vip.xml</include>
</policy>
```

The high-level policy file does not depict which server is currently serving which subnet, and where the configuration files are located, etc. This type of unnecessary information is part of the “incidental complexity” that closures are designed to avoid. By excluding nonessential and architecture-specific information, the high-level policy can achieve a high level of reusability.

The low-level policy file contains nearly the same information as the DHCP/DNS configuration files, but it also contains the running state of the peer-peer system. For example, the following code is a part of the low-level policy. The ‘auth’ attribute records the current “boss” in charge of this segment. The ‘failover’ attribute shows that the failover is on. This protocol distinguishes between owners of information at a relatively fine grain.

```
<closure ts="1136491620">
  <dns>
    <ip>192.168.0.100/24</ip>
  </dns>
  ...
  <subnet-segments>
    <!-- The auth attribute hold the
                                current owner of this subnet. -->
    <subnet physical="192.168.0.0"
      authMAC="00:02:3F:1F:9C:88"
      auth="192.168.0.21/24"
      failover="on">
      <id>192.168.0.0</id>
      <netmask>255.255.255.0</netmask>
      <max-lease-time>51000
    </max-lease-time>
    <pool>
      ...
    </pool>
  </closure>
```

When changes are needed, such as changing the range of available IP addresses, or IP renumbering [11], IP address closure can ease the job of an administrator. Currently renumbering is very labor-intensive and requires a series of carefully orchestrated steps. Given a change in policy, the closure could in principle take over this orchestration and accomplish the renumbering with minimal outside help. This includes validating that the renumbering is possible, and actually performing the renumbering once it is proved to be valid, leading the human administrator through a series of foolproof steps.

Implementation Details

We implemented a prototype of the IP address closure using the ISC DHCP [8] and BIND [7] software. The gossip protocol is built on TCP, and policy content is encoded in XML. The information is managed using Berkeley DB XML by sleepycat [21]. Our test environment consists of ten PCs running Linux. They are separated into four IP subnets connected by PCs configured as routers.

To implement self-bootstrapping, we extended the function of DHCP client and implemented the logic

shown in Figure 5. The PIP box is pre-installed with the modified version of the ISC DHCP v3.0.2 package. When a PIP is booted, the Ethernet interface is configured to obtain its IP via DHCP. If an IP is obtained from fellow PIPs, the booting PIP will launch the gossiping process; otherwise, the self-bootstrapping process starts. The gossiping protocol is implemented as a pull-only peer-to-peer gossiping application. The transformation from a low-level policy to the actual configuration file utilizes XSLT [22] technology.

In our current setting, the size of contents in a PIP is around 4KB. The cloning of whole contents (from one PIP to a newly installed one) happens in about one second. We set the interval between two pull operations to be 20 seconds. Because of the size of our testing environment, the propagation delay is bounded to 20 seconds as well. Propagation delay is affected by both the frequency of pulling and the number of neighbors each PIP has. In our setting, it is safe for a PIP to have 10 neighbors. It will be interesting to validate this light-weight protocol in a real large-scale enterprise environment, and discover a range of optimal number of neighbors that each PIP should have.

The current capabilities of this prototype are bootstrapping, the dissemination of high-level policy and proposal through a P2P network, high-level to low-level policy translation, and automatic DHCP server configuration update (ISC DHCP only). Future self-managing features of an IP address closure (yet to be implemented) include policy-environment conflict detection, IP address pool shortage warning and auto-allocation. We achieved many of our goals in this prototype: to validate the feasibility of (1) self-bootstrapping, and (2) realizing a distributed configuration based on a high-level policy to provide a robust IP infrastructure.

Autonomics and System Administration

The popular vision of “autonomic computing” (or “self-managing systems”) is that there will be no system administrators and systems will manage themselves. This vision is inaccurate and naive. Before an autonomic system becomes functional, much initial setup work must be completed by administrators. After the system is successfully configured into a functioning state, the system is monitored by both self-managing components and system administrators. If a problem occurs and it is beyond the self-healing ability of the autonomic system to correct itself, administrators must take over and restore the system to a functional state.

A rather obvious property of autonomic systems is also their most major pitfall. Current autonomic systems can only cope with predictable failure modes. If something unpredictable happens, a human is required to intervene and take appropriate action. The system can “learn” (postmortem) what it should have done, but cannot cope with the new problem without guidance and help.

Here there is a major and unexpected pitfall for the system administrator. The problems with which an autonomic system cannot cope are also problems that may stump the experienced system administrator. The autonomic system is best thought of as a “junior system administrator” armed with a set of “best practice” scripts that can solve most problems. When a problem does not fit any known description, then by nature, *advanced intervention* is needed. The system administrator who can cope with problems of this nature must be *better* trained than many current system administrators, with an emphasis on efficient and rational troubleshooting. But how (in the context of self-managing closures) does the system administrator achieve this high level of training, when the closure is trying to take control away, and isolate the system administrator from the behavior of the system? One cannot both train and isolate the system administrator. This is a major quandary in the design of autonomic systems: how will the administrator achieve the level of knowledge required to cope with contingencies?

Administering the IP Address Closure

We use the IP address closure as an example to discuss the impact of similar autonomic systems upon system administrators. The system administrators delegate some low level decisions to the closure. Thus, they can focus on the larger picture of IP address assignment schemes. The IP address closure relieves system administrators from the job of backing up policies, because PIPs clone policies from one another and are in essence self-preserving.

However, in no way is the system administrator redundant in the IP address closure. The closure cannot control or define the physical connectivity between devices, or guarantee the architecture of physical or virtual subnets. The system administrator has a permanent role in matching the physical architecture of the network with policies, and in intervening when the closure discovers a mismatch between the physical network and desired operating characteristics.

Another unavoidable role is that of bootstrapping the system from a non-functional state to a self-managing state. In our closure, this is accomplished by physical moves of devices. This eliminates common human errors in copying configurations and makes the bootstrapping protocol more or less foolproof, but requires a basic understanding of how the PIPs self-replicate.

Lessons Learned

The PIPs show us something fundamental about the ongoing relationship between system administrators and autonomic elements. The administrator is far from obsolete, but also somewhat removed from the day-to-day management tasks that were formerly part of the job. The system administrator becomes a crafter of policies, rather than just a troubleshooter. Far from being less skilled, the system administrator of the closure system actually needs a higher level of sophistication to deal with unexpected problems.

The changing role of system administrators includes the deployment and bootstrapping of autonomic systems. Each autonomic system has a set of preconditions that must be met before it can function as designed. System administrators must maintain the appropriate environment for the autonomic system. Before the autonomic mechanisms are implemented from top to bottom, each autonomic system must rely on human administrators to cope with the bottom layers. Although these autonomic systems provide self-configuration features, deployment and bootstrapping are unavoidable and uniquely human tasks.

The role of system administrators also include tuning and validation of the new autonomic systems. Many autonomic systems contain heuristics that require the collection and analysis of real production data. Before the system is tuned, system administrators may have to manage the system manually. After the system is tuned, it must be validated to make sure that it is configured as desired.

Administrators also must intervene when a problem cannot be handled by an autonomic system. This poses new challenges to autonomic systems and their users. Unlike current practice in which the administrators have absolute control, system administrators must turn off certain parts of the automated process and take over, just like taking over from automobile cruise control or auto-piloting. The responsibilities must be well-defined and documented. The switching process between autonomic and manual management must be well documented and practiced.

An autonomic system itself is more complex than a corresponding traditional system. For example, in configuring our PIPs, one must describe their operating environment and policies in great detail, giving information that would not be known by hosts in a non-autonomic DHCP/DNS infrastructure, such as the locations of routers on foreign subnets. This extra configuration, however, pays off when the resulting fabric of servers relieves one from routine tasks such as rebuilding servers or propagating changes to the network.

One primary obstacle to acceptance of autonomic solutions is trust [1, 10]. People often do not trust that machines can handle tasks reliably, when humans will lose control. In truth, autonomic solutions are “assistants” rather than masters; the fabric of management still contains both machines and humans. This paradigm is especially necessary at the lower levels, where human assistance is required. The human administrators can use help in implementing complex processes. One goal for autonomic systems is to automate IT service and resource management best practices [4]. Automating these best practices can best gain the trust of the management and administrators. Further, autonomic assistants can help humans track the state of a task. Our problem domain is close to the hardware; so close that a human element cannot be

avoided to serve as the “hands and feet” of the system. Accountability issues are also unavoidable. Who should be responsible if a system is not tuned well and does not meet the specific requirement of a site and cause downtime?

Our brief discussion of the changing role of system administrator may seem daunting, but the job is in no danger of extinction. Current closures require extensive bootstrapping and handling of contingencies, and require monitoring by a highly skilled system administrator. In fact, management of autonomic systems seems to elevate the profession in several ways:

1. by requiring a high level of system administration expertise.
2. by redefining the role of system administrator as someone who directly interacts with policy.
3. by providing (through interaction with policy) upward mobility to management positions.
4. by providing a much needed human interface between autonomic elements and upper management.

Exception Handling

The effectiveness of an autonomic solution depends upon the efficiency with which humans can communicate with it. Our PIPs cannot solve all problems, so that their ability to effectively communicate problems to humans is crucial to their success.

A key feature of any autonomic system is how it handles cases in which self-management does not resolve an issue. The goal of exception handling in autonomic systems is to report any violations of policy and resolve them. For example, in the IP address closure, suppose an interface is declared to have a fixed IP address. When an administrator activates that interface, if this host is not physically located on the same subnet as it should be, the interface might get an IP address from DHCP on a different subnet than desired. In this scenario, no obvious error is generated. However, this is a clear exception to the policy. In this example, the root cause is that the interface is not connected physically to the proper subnet. We may choose to correct the root cause by physically moving the interface or perhaps setting up a VLAN to simulate physical rewiring. Alternatively, we could choose to reject the policy on the grounds that it is not implementable. This constitutes a form of *exception* in the closure.

The concept of exception has been widely used in the computer science field. Exceptions have been used mostly to express a predictable abnormal scenario that can be categorized in advance. Researchers have explored exception handling mechanisms in both intra-component and inter-component situations. For example, Romanovsky divided exception handling into local error detection in one component and coordinated action level handling among components [19]. Here we focus on handling of unexpected exceptions, or exceptions with unknown causes.

Because a closure defines observable behaviors, it is natural to define the possible exceptions raised by this closure also in terms of observable behaviors, just like the symptoms of patients. However, unlike the policy file, the more detailed the exception, the more helpful the information. A simple exception containing no extra information will almost definitely require a human to troubleshoot the problem.

The exceptions that a closure could raise can be divided into two categories:

- Exceptions that cannot be handled by this closure. The cause of the exception is out of the scope of control of a certain closure. i.e., self-healing does not function properly in one situation. For example, in the Apache closure, if the file system is corrupted, the closure cannot possibly work properly, thus an exception must be thrown. Sometimes, the reason for the exception is unclear. Thus a generic exception might be raised.
- Exceptions that may be handled by this closure but that the closure chooses not to handle. Rather, the closure wants the exception to be handled by other closures.

For example, in the Apache closure, if the server is unstable, the closure may choose to restart the server. Or, if the closure decides that a better way to handle this is to reboot the whole host machine, it may raise an exception and let another entity handle it (such as a host closure).

A special type of exception is related to human input. When the closure discovers that the intention of the administrator is unclear, or it encounters a condition where more human input is needed, it should raise an exception to request more information, instead of relying upon itself.

Exceptions can be handled by other closures or human administrators. Since we cannot wait to have closures to be built on all the layers and switched on at once, it is necessary to have a way for closures to request services from other non-closure systems or human administrators. In the exception-handling process, after the event causing the exception is resolved, the closure can be contacted manually or programmatically to continue its work. In a true autonomic system, most exceptions should be handled by a program, rather than by a human administrator.

Conclusions and Future Work

We propose an “IP address closure,” a self-managing IP management infrastructure providing DHCP and DNS services. The IP address closure mimics the best practices that administrators discovered in practice, and automates them through the coordination among Peered IP management nodes (PIPs). Thus, the IP address closure is designed to gain the trust of the system administrators to assist with their work.

The task of making low-level systems self-managing still requires solving many open problems. The

key problem for IP management is to maintain an effective interface between the fabric and its human counterparts. Human administrators are not obsolete, and they are still critical because autonomic systems cannot escape exception problems due to physical limits upon architecture. However, designing policies and resolving exceptions might require a new set of skills for existing administrators. The policy still depends upon architecture.

The most complex and challenging problem is that of planning for safety in very complex changes. When policies change, there are often safe and unsafe ways to transition between policies, where an unsafe transition is one that temporarily exposes a security risk.

Another related problem is how to make the lower layers (routing and switching) self-managed in a similar way. These layers suffer from the same “bootstrap problem” that we observe for IP address management; the management fabric has to use what it manages for its own sustenance, and cannot do that until it manages that fabric. The simple solution of managing routing via an out-of-band management network may not be cost-effective for many sites.

Clearly, there are many issues to explore. If there is a single most important contribution of this paper, it is that the closure idea is possible at the IP layer, and that – even with bootstrapping difficulties – self-managing fabrics can function near the physical layer of a network, provided that there is a carefully orchestrated relationship between the self-managing fabric and its human partners.

The role of administrators in the autonomic era has already changed. Instead of being obsolete, autonomic systems challenge system administrators to obtain a higher level of expertise, including knowledge of policy design and architecture, tuning, and troubleshooting. At the same time, autonomic systems elevate the system administration profession and shorten the distance between management and system administration through the common language of policy-based interfaces. Some system administration jobs may be lost to autonomic systems, but those that remain may well enjoy better advancement opportunities, as well as increased respect and recognition for the profession.

Author Biographies

Ning Wu is pursuing his Ph.D. at Tufts University. His research interests are in system management, autonomic computing, system integration, and P2P systems. Before studying at Tufts, he had worked as an engineer for Genuity and Level 3 Communications Inc. He received an M.S. from State University of New York at Albany, an M.E. from East China Institute of Computer Technology, and a B.S. from Southeast University in China. Ning can be reached via email at ningwu@cs.tufts.edu.

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from M.I.T. in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Computer Science at Tufts. Prof. Couch is the author of several software systems for visualization and system administration, including Seecube(1987), Seeplex(1990), Slink(1996), Distr(1997), and Babble(2000). He can be reached by surface mail at the Department of Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as couch@cs.tufts.edu.

Bibliography

- [1] Chan, Hoi, Alla Segal, Bill Arnold, and Ian Whalley, "How can we trust an autonomic system to make the best decision?" *2nd International Conference on Autonomic Computing (ICAC 2005)*, pp. 351-352, 2005.
- [2] Cisco Systems, *Cisco CNS Network Registrar Users Guide, Software Release 6.1*, 2004.
- [3] Couch, Alva, John Hart, Elizabeth G. Idhaw, and Dominic Kallas, "Seeking closure in an open world: A behavioral agent approach to configuration management," *Proceedings of the 17th Conference on Systems Administration (LISA 2003)*, pages 125-148, 2003.
- [4] Ganek, A. G. and T. A. Corbi, "The dawning of the autonomic computing era," *IBM Systems Journal*, Vol. 42, Num. 1, pp. 5-18, 2003.
- [5] IBM, *An architectural blueprint for autonomic computing*, IBM white paper, April, 2003.
- [6] International Network Services, *IPControl*, <http://www.ins.com/software/ipcontrol.asp>.
- [7] Internet Systems Consortium, Inc., *ISC BIND*, <http://www.isc.org/index.pl?sw/bind/>.
- [8] Internet Systems Consortium, Inc., *ISC Dynamic Host Configuration Protocol (DHCP)*, <http://www.isc.org/index.pl?sw/dhcp/>.
- [9] Keller, A., J. Hellerstein, J.L. Wolf, K. Wu, and V. Krishnan, "The champs system: Change management with planning and scheduling," *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, Kluwer Academic Publishers, April, 2004.
- [10] Kephart, Jeffrey O. and David M. Chess, "The vision of autonomic computing," *IEEE Computer magazine*, January, 2003.
- [11] Limoncelli, Tom, Tom Reingold, Ravi Narayan, and Ralph Loura, "Creating a network for lucent bell labs research south," *Proceedings of the 11th Conference on Systems Administration (LISA 1997)*, pp. 123-140, 1997.
- [12] Logan, Mark, Matthias Felleisen, and David Blank-Edelman, "Environmental acquisition in network management," *Proceedings of the 16th Conference on Systems Administration (LISA 2002)*, pp. 175-184, 2002.
- [13] Lucent, *Lucent network management software for enterprises*.
- [14] Melcher, Brian and Bradley Mitchell, "Towards an autonomic framework: Self-configuring network services and developing autonomic applications," *Intel Technology Journal*, Vol. 8, Num. 4, Nov., 2004.
- [15] Mockapetris, P., "Domain names – concepts and facilities," *RFC 1034*, 1987.
- [16] Mockapetris, P., "Domain names – implementation and specification," *RFC 1035*, 1987.
- [17] Network Working Group, *DHCP failover protocol*, 2003, <http://www3.ietf.org/proceedings/04mar/I-D/draft-ietf-dhc-failover-12.txt>.
- [18] Rhea, Sean, Brighten Godfrey, Brad Karp, John Kubiawicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu, "OpenDHT: A public DHT service and its uses," *Proceedings of ACM SIGCOMM 2005*, 2005.
- [19] Romanovsky, A., "Exception handling in component-based system development," *The 15th Int. Computer Software and Application Conference, COMPSAC 2001*, 2001.
- [20] Schwartzberg, Steven and Alva Couch, "Experience in implementing a web service closure," *Proceedings of the 18th Conference on Systems Administration (LISA 2004)*, 2004.
- [21] Sleepycat Software, *Berkeley DB XML*, <http://www.sleepycat.com/products/bdbxml.html>.
- [22] W3C, *XSL Transformations (XSLT) Version 1.0*, <http://www.w3.org/TR/xslt>.
- [23] W3C, *Web services architecture*, 2004, <http://www.w3.org/TR/ws-arch/>.

Modeling Next Generation Configuration Management Tools

Mark Burgess – Oslo University College
Alva Couch – Tufts University

ABSTRACT

There are several current theoretical models used to discuss configuration management, including aspects, closures, and promises. We examine how these models relate to one another, and develop a overall theoretical framework within which to discuss configuration management solutions. We apply this framework to classify the capabilities of current tools, and develop requirements for the next generation of configuration management tools.

Introduction

Configuration management is the process of constraining the behavior of a network of machines so that each machine's behavior conforms to predefined policies and guidelines and accomplishes predetermined business objectives. Configuration management would be easy if objectives did not change, the number of machines in a network were small, machines were simple in structure, machines were identical, machines did not fail, and no unauthorized parties could alter behavior. Unfortunately, objectives change, networks are large, machines are complex, machines differ, machines fail, and new security holes appear each week, making configuration management a costly part of administering any large network.

The overall goal of configuration management as a practice is to maximize the extent to which systems conform to predetermined expectations, while minimizing the cost of keeping the network's behavior within predetermined guidelines. There are many strategies for accomplishing configuration management, from manually making changes to using powerful and comprehensive software tools to assert, enforce, or monitor configuration changes. Each approach has loyal advocates who consider their approach superior to others, leading to spirited debates in the LISA Configuration Management Workshop.¹

At the present time, configuration management remains one of the most controversial aspects of system administration. Approaches and tools abound, each with faithful adherents whose dedication to the approach borders on religious fervor [1, 3, 4, 5, 6, 13, 20, 21, 22, 23, 24, 27, 29, 30]. Understanding the key differences between approaches has proven difficult, and many practitioners have asked why it seems so necessary to “re-invent the wheel” [27] in creating completely new configuration management tools from scratch. Many

¹What began as the CFEngine Workshop at LISA 2001 inspired many discussions and was more appropriately renamed the Configuration Management Workshop by Paul Anderson shortly afterward.

tools seek to enable new collaboration methods(e.g., [23, 27]) to enable sharing of work. So far, collaboration seems to be the exception rather than the rule.²

Are authors of new tools really contributing anything new? Why has it proven so difficult to apply configuration management tools to new sites? In trying to answer this question, it has proven difficult to get beyond issues of personal taste and understand *why* there are so many approaches, and what drives each approach. Part of the reason for this is that there has been no coherent language with which to compare and contrast strategies both precisely and fairly. Without this language, advocates of particular approaches seem like zealots; with this language, the reasons behind their thinking can become clear.

The Quandary of Cost

One reason for the diversity of approaches is that the least costly strategy for configuration management is often determined by the nature of the site being managed and its mission [19]. There is, for example, a profound difference between the best configuration management strategy for an academic research lab and for a bank. “Tight” sites such as banks require much more disciplined and expensive strategies than academic research labs, because the cost of downtime is much higher in a bank than in a research lab [26].

To better understand the sources of cost, there have been several tries at creating theoretical models of configuration management. *Closures* attempt to encapsulate parts of network function in black boxes, to reduce configuration management complexity and cost [15, 16], while *promises* model the way autonomous parts of a network exchange information and commit to certain behaviors [10, 11, 12, 14], to allow networks and computers to become more self-

²AC: At a configuration management “birds of a feather” session at LISA 2003, an informal poll was taken concerning the number of people in the room using *other* authors’ configuration management tools. Of the attendees present, excepting users of CFEngine, everyone had written custom tools for the task, and the only user of each tool in the room at the time was its author.

managing and self-sufficient. We present *aspects* as a way of describing the dependencies and constraints that plague configuration management and increase management cost [2]. The diversity and disparity of the contributions has been a hindrance to a feeling of progress in the field. Why are there so many different ways to think about the same basic problem?

In this paper, we discuss the relationships between current models of configuration management with several goals in mind. We define the terms and concepts in each model precisely, and show how they relate to concepts and terms in other models. This leads to an overall theoretical picture of configuration management based upon the union of concepts. This unified theory suggests and clarifies challenges to be addressed by the next generation of configuration management tools.

The plan for the paper is as follows: we introduce the concept of an aspect to capture a configuration management ‘unit of planning.’ We can think of this as a *requirement*. We then discuss how such aspects can be reliably implemented. This takes us to the concepts of closures and promises. However, there is an obstacle: it is far from clear that we have the authority to require anything of a distributed system, so we must transform a description of requirements into a description based on agreed compliance, or promises. Reviewing briefly the concept of service-oriented computing we show that, if we express aspects and closures in terms of ‘promises,’ then all of our results apply regardless of whether they are implemented as granted services or as authoritative control scripts. This abstraction makes our theory completely general. We finish by indicating how convergent operational semantics can be expressed as promises, thus completing the picture from high to low level.³

How Expensive Could It Be?

It helps to understand from whence configuration management costs arise. The cost of configuration management includes the costs of planning, deploying machines, deploying changes, and troubleshooting changes. Planning includes determining desired behaviors and how to accomplish them. Deployment consists of creating machines with a known initial configuration, to which configuration changes can be applied later. Changes are deployed by modifying machine configurations, network-wide, and changes often cause problems that must be investigated through troubleshooting.

Some of these costs are fixed and difficult to control, while others are somewhat under the control of the system administrator. Planning costs the same amount of staff time regardless of how one decides to

³*MB+AC*: for the reader’s amusement we have left our (often wry) commentary to one another as bonus material to the director’s cut of this paper.

manage systems, but deployment costs vary based upon whether the deployment is accomplished automatically or manually. The cost of troubleshooting shows the greatest variability and greatest opportunity for savings. It can be argued that the cost of troubleshooting is the *sum* of staff cost for repairing the problem and staff time and revenue lost *due* to the outage [26]. This observation makes troubleshooting a dominant factor in overall cost of ownership.

Constraints, Dependencies, and Preconditions

One core problem in configuration management is that accomplishing changes is often nontrivial. Often, when a change is made, “something breaks” [28], and troubleshooting is required to determine the cause. For example, installing a new version of a dynamic library has the potential to cause every program that loads that library to stop working properly. In Microsoft Windows, program installers can modify the registry entries of other programs, either intentionally or maliciously, so that installing a new program can lead to seemingly unrelated failures [31]. Programs often invoke other programs. For example, installing an inappropriate version of GhostScript can prevent Xfig from generating Postscript figures.

There are several ways that different authors describe the above situation in words. One can say that “there is a *dependency* between Xfig and the version of GhostScript” or “there is a *constraint* that the versions of GhostScript and Xfig must match.” These are equivalent statements. A *precondition* [18, 24] is another name for a dependency; one could say that “a *precondition* for Xfig to function properly is that the appropriate version of GhostScript is installed.” The difference between a precondition and a dependency is that a precondition describes relationships between events or occurrences in time, while a dependency or constraint describes relationships between subsystem states.

But the above situation is the easy case. Often, we do not know (or perhaps forget) the dependencies or constraints that must be satisfied. In making changes, it is possible to put systems into states whose behavior is unknown or unverified. Usually, this is because the system is in a different state than we believe it to have, either when applying a configuration change or when trying to use a program. We can say a failure is due to a *hidden dependency* or a *hidden constraint*, or that the success of a command requires (or prohibits) a *latent precondition* [24].

Aspects

We begin our story by proposing a definition of configuration management based upon *aspects*.⁴ Our definition of an aspect differs somewhat from that of Anderson [2], who defines it as “a part of configuration specified by one human person or administrator,”

⁴*MB*: on hearing about Promise Theory, Alva turned crimson and sang – “Mark’s tongues in aspects.”

but we agree with the spirit of his remarks. Instead, we define an aspect of configuration as a bundle of configuration information whose values must be coordinated to satisfy some known set of a priori constraints.

Aspects Are Required Characteristics

Our extended definition has more precise mathematical properties than Anderson's definition, but satisfies the spirit of the original definition; typically Anderson's hypothetical "one person" will be charged with deciding the values for a single aspect.

We pursue this course not so much because aspects are interesting in and of themselves, but because they provide the "glue" and intermediate representation that allows us to discuss the similarities between the seemingly very different concepts of "closures" and "promises," i.e., the concepts we need to actually implement configuration changes. They also better represent the way in which administrators plan and think about distributed management.

We begin with some definitions. It is necessary to understand precisely what we mean by a "configuration parameter" or a "constraint upon configuration parameters" before we can characterize the problem of configuration management more accurately. This may seem overly precise, until one considers that the lack of precise definitions has historically made it difficult to compare configuration management approaches, because authors have utilized differing terms to describe similar concepts.

Definition 1: Configuration parameter A configuration parameter is a unit of configuration information. It can be manipulated by use of specified get and set methods, where get returns the parameter's value and set specifies a new value.

A parameter's location within the system is not important, we refer to it indirectly, i.e., by a method or access service which conceals that specific location. The value of a parameter might be anything from a single scalar value to the contents of a hierarchy of files located somewhere within the filesystem.

We now want to talk about aspect 'types.'

Definition 2: Type of a parameter The type of a configuration parameter p is a label identifiable with the domain of possible values that the parameter can assume, which we notate as D_p .

Note that a type is a shorthand for a set of options D_p . Try not to think of an aspect type as a primitive data-type, e.g., like "string" and "integer"; rather think of "parameter set 1" and "parameter set 2" in a system specification, i.e., different configuration concerns. Types have the character of database schemas or XML schemas for configuration parameters.

Definition 3: Parameter constraint A single constraint on a configuration parameter p may be expressed in two equivalent ways:

1. As a restricted set of values, i.e., a subset V_p of the domain D_p of its allowable values.

2. As a set of rules R_p that indirectly define the contents of V_p .

The reader may be confused by this abstract specification of a rather mundane thing. When we make a constraint upon a parameter, e.g., "the hard disk must contain more than 4 GB of space," we are in actuality selecting a subset of hard disks that meet the criterion. By thinking of this set, rather than the rule or formula that defines it, we can avoid messy notation and clarify the concept.

Definition 4: Parameter set constraint For a parameter set A , we can think of the constraints on the set A as being specified in two ways:

1. As the union of the rules R_p for values of parameters $p \in A$.
2. As a set of allowable tuples of values $T_A \subseteq \prod_{p \in A} D_p$ where \prod denotes cross product.

In other words, the constraints on a set of things are some subset of the ordered tuples of parameter values, or alternatively, some set of rules that determine those tuples.

Example 1 It is common for sets of parameters to have constraints between parameter values. The hostname declared for the web server is usually the same name as the name of the physical host running the server. This is a form of tuple constraint.

A constraint is a specification of a subset of the possible parameter values that we particularly require. Defining rules R_A expresses that, for the parameters P_A , we are disallowing some tuples of values and are left with a smaller set of tuples T_A that are suitable. This set may be defined by enumerating possible tuples, or by abstract rules, but the effect is the same: to limit the set of allowable values.

An aspect, then, is a logical grouping (schema) of such parameters whose values are a characteristic of the system that we are trying to manage.

Definition 5: Aspect An aspect A is a pair $\langle P_A, C_A \rangle$, where P_A is a set of configuration parameters and C_A is a set of constraints limiting the values of those parameters. C_A may be expressed as either a ruleset or an enumeration of tuples.

If a parameter p is part of an aspect, values of the parameter p must conform to the constraints V_A for the overall aspect (specified as a tuple space). If the value of one parameter p is changed, we must choose a new value $v \in V_A$ (the set of allowable tuples), so that $v(p)$ has the value we desire, while all other parameters are adjusted so that the value of the aspect v remains a member of the constrained set V_A of allowable values.

Example 2 Suppose that two data values are required to have the same value, but are stored in different places and accessed via different means. They are different parameters, but can be

considered to be members of the same aspect, bound together by the aspect constraint of “value identity.” For example, a web server must be configured to answer requests for a numeric IP address that happens to agree with the IP address of the machine running the server.

If we have two parameters whose constraint is that they must have the same value at all times, then set for one must set the same value for the other, using mechanisms of traditional aspect-oriented programming.

Example 3 Consider the number of times the hostname of the current host appears inside files in etc. Under our definition, each occurrence is a separate parameter, but the aspect “hostname” embodies all of them, and setting the hostname as an aspect should modify all occurrences of the hostname everywhere it might appear in configuration files. There are many aspects with this quality, of one value stored in many places.

Again, we have a relationship between setting one parameter and setting several others. Aspects may be more subtle than identity, though.

Example 4 Consider an aspect dealing with hostname in a local-area network. The hostname of each host must be unique, so we make this property an aspect of the local-area network. Setting the hostname of one specific host to an already-assigned name would require us to set the already-assigned host’s name to something different.

Aspect constraints can be much more complex than this:

Example 5 Consider an aspect for installing software packages. This aspect has constraints for determining when a package can be considered to work, in terms of dependency packages that must be installed beforehand.

Aspects can even honor dependencies inside a single software package.

Example 6 In the aspect called “web service,” there are specific requirements and limitations on which modules can be installed in Apache, due to interoperability limits.

In general, we can model the dependencies, requirements, and documentation of a network as a mesh of overlapping, inter-dependent aspects. Overlaps will be a problem, but we shall solve this matter by reducing aspects to networks of “promises.” Our definition of an aspect is very similar to that of a promise [14], but at a higher level, and indeed this is no accident. We shall be returning to the reason for this in later sections.

Properties of Aspects

Having introduced aspects, the concept of parameter becomes somewhat redundant: we can meaningfully converse in terms of aspects alone.

Proposition 1: Any single parameter p is also an aspect $\langle p, D_p \rangle$.

Proof 1 The type of a parameter by definition corresponds to a set of domain values, so the result is trivial.

Proposition 2: Any set of parameters is also an aspect, with the set of constraints that is the union of their individual constraints/types.

Proof 2 Again, this is obvious from the definition.

Aspects consisting of only type information are rather dull; to make life interesting, we must include constraints about how parameters interoperate or must be related. We can do this most straightforwardly via composition:

Lemma 1: Aspect composition A union of aspects is an aspect.

Proof 3 Let $\langle P_A, R_A \rangle$ represent one aspect A (expressed as a set of parameters P_A and a set of constraint rules R_A) and let $\langle P_B, R_B \rangle$ be another aspect expressed in the same fashion. The lemma follows trivially from

$$\begin{aligned} A \cup B &= \langle P_A, R_A \rangle \cup \langle P_B, R_B \rangle \\ &= \langle P_A \cup P_B, R_A \cup R_B \rangle. \end{aligned}$$

The union of the sets of constraints is a larger (and perhaps more restrictive) set of constraints.

In other words, to make a union of two aspects, take the union of their parameter and constraint sets. Naturally, the behavior of a larger set of constraints is more constrained than a smaller number; as we make successive unions of aspects we arrive at a system with completely determined behavior at top level.

The duality between constraint rules and allowable value sets may seem curious to the reader. A rule $r \in R_A$ limits an aspect, which means that the larger R_A is, the smaller the set of allowable values V_A becomes. The same apparent strangeness occurs in object-oriented programming, where adding constraints to a subclass (via inheritance) limits the number of instances that can be considered members of that subclass, compared to the instances that are members of the parent class. Increasing constraints limits the number of acceptable values. Decreasing the number of constraints increases the number of acceptable values. This duality and contravariance between constraints and instances will be exploited in several ways in the rest of the paper.

Definition 6: Value of an aspect The value v_A of an aspect A is a function from parameters within the aspect to values of those parameters, so that $v_A(p)$ represents the current value of parameter $p \in P_A$.

Again, this is a simple concept. A value is simply a tuple v_A where fields $v_A(p)$ conform to all constraints of the aspect.

Hard and Soft Constraints

Note that constraints on values take two forms: “hard” and “soft.” A “hard” constraint is one whose violation also violates physical law or the preconditions of a software or hardware subsystem. A “soft” constraint is a matter of policy or personal taste or choice.

A “hard aspect” contains only hard constraints:

Definition 7: Hard aspect *A hard aspect is one in which all constraints reflect physical limitations of the configured device and/or its software.*

For example, not using existing partitions in a partition map would lead to a non-functional system. Hard aspects arise both from documentation (of which values “should” work) and direct experience (of what works and does not work).

A “soft aspect” is one that we impose as a matter of policy, even though no physical laws are broken in its absence.

Definition 8: Soft aspect *A soft aspect is one in which all constraints are elective and do not reflect actual physical limitations. We might also call this a policy aspect.*

For example, the place we actually install the web server software is a “soft aspect” of the web service hierarchy; there are no physical reasons we cannot install it anywhere we wish (provided that partitions are large enough, which is a hard aspect!).

If we consider that the parameter that we are setting is itself an aspect, and the value we are asserting is a constraint within a new aspect containing that parameter, then our desires for a host’s configuration can all be expressed in terms of aspect composition. Indeed,

Proposition 3: Configuration is an aspect *The entire configuration of a host or network can be thought of as the value of a composition of hard and soft aspects, including physical limits, policy choices, and arbitrary choices.*

An aspect generalizes and embraces related alternatives, i.e., one choice is available for each parameter in an actual configuration, whereas an aspect may provide alternatives. We can therefore arrive at a configuration by imposing a sequence of increasingly demanding constraints, from hardware and software limits, tempered by policy decisions, all the way to individual choices that may not matter.

The key idea of aspects is that it is an easy and straightforward way to encapsulate relationships between parameters and subsystems. While a parameter corresponds to a single configuration item, an aspect binds several together with a shared meaning, that might be either localized or distributed. In this respect, aspects will turn out to be related to *roles* in promise theory, which we will also discuss.

Managing Aspects

The concept of an aspect is a compelling mirror of design practice. Implementing configuration management,

one must constantly conform to a series of practicality and policy constraints. These constraints commonly *overlap*, making configuration management a constraint satisfaction problem [25]. Worse, the constraints of an aspect may not be known, and we sometimes must make guesses about their nature. We can thus rethink configuration management as a problem of managing aspects.

Clearly, aspects are a mechanism for specifying the *requirements* for a functional system. But there is a presumption here – namely that we can actually require anything at all of a system. As computers and devices become increasingly personalized, a configuration planner becomes increasingly powerless to control autonomous devices; this is an issue which we are forced to confront.⁵ An aspect specification is completely free of assumptions about how it will actually be managed, as a physical entity. This conceptual decoupling allows us compile the high level concept into some kind of lower level language – and this brings us to discuss closures and promises below.

Many aspect constraints are simple value choices. We can conform to these constraints most easily by storing the (replicated) specification in a database or file, and replicating the information into several files via a “generative” [18] or template based configuration management strategy, e.g., like LCFG [1, 3, 20].

Since the definition of a parameter arises from the ability to get and set it, two parameters are identical if they are defined by exactly the same get and set methods. In case of overlaps, it is important to know whether values for two overlapping aspects are reasonable:

Definition 9: Coordinated aspects *Two aspects are mutually coordinated iff they agree to share the possible values of aspect parameters.*

In promise theory one has the notion of a coordination promise as a primitive construction to handle scenarios like this. Compiling aspects into promises will allow us to keep track of the logic of these complexities.

The most difficult aspects to manage are those with “distributed constraints.” While a “local” aspect involves one machine, a “distributed” aspect involves some group of machines and their interactions. These have proven difficult to manage in several ways. First, there is a need for *coordination* whenever part of an aspect must change on one host in an aspect group.

An example of a distributed aspect is a client-server relationship. In this relationship, a client has an aspect that identifies the server, while the server has an aspect that defines the service. The union of these aspects and a port-number aspect describes a *binding* between server and client.

⁵MB: There is an important crossroads here. As we move towards a service-oriented picture of autonomously managed services, we move into a realm of having no authority to require anything of a server. Thus we must eventually move away from the idea that we are in control, to a view of encouraging voluntary cooperation. This step is taken by reinterpreting aspects in terms of promises [14, 10, 12].

Proposition 4: Bindings are distributed aspects
All service bindings of a client are distributed aspects with both client-side and server-side components.

Example 7 *For a more complex distributed aspect, consider the inherent coupling between DHCP and DNS. If a host is in DHCP, then its MAC address maps to a particular IP address, and if it is in DNS, then its IP address is mapped to a name. Thus the triple (hostname, IP address, MAC address) is a distributed aspect spanning the host itself, the DNS server, and the DHCP server. Once the value of that aspect is defined, it constrains values in all three domains and, by overlap, constrains contents of other configuration files whose aspects overlap. It is often considered good form to place a record for the hostname of a host into /etc/hosts; this would happen because the hostname aspect (on the local host) must be coordinated with the DNS/DHCP aspect (on the distributed network service layer).*

State of the Art

At most sites, distributed aspects are maintained and updated by hand, by distributing policies for *local* aspect control. For example, the policy writer must insure *manually* that the DNS server listed in /etc/resolv.conf is actually a DNS server, and that the zones of that server contain the appropriate SOA records for it to be an authority for the zones for which it is intended to be authoritative.

This difficulty in coordinating distributed aspects is also largely responsible for the incorrect belief that centralized coordination is necessary for effective configuration management. Tools such as LCFG [1, 3], BCFG [20], Puppet, Arusha [23], and others manage distributed aspects through centralized coordination. Mostly this is accomplished by storing values in a single data structure that can be checked on a central server for consistency. The strength of generative configuration management is that identity relationships among aspects (where several parameters must have precisely the same value) are addressed by generating multiple files from the same hierarchy of values, thus solving the aspect consistency problem implicitly.

To our knowledge, with one exception, none of the data models of production configuration management tools are explicitly aware of aspect constraints; they manage aspects by constructing some values as functions of others. Thus the kinds of aspects a typical tool can express are rather simple ones, where there is a functional relationship between the choice of value for one parameter and choices for values of others. The exception is Anderson's prototype implementation using SmartFrog [3], which dynamically computes aspect constraints and chooses among the resulting values. CFengine [4], on the other hand, is explicitly constraint

oriented at the low level, but does not cope well with high level or aspect-level constraints.

One problem with contemporary mechanisms is that they take a lot of human labour to set up, and require that the centralized hosts generate configurations by acquiring and maintaining rather intimate knowledge of the hosts that they manage. In order to manage a distributed aspect, a central server must control *both sides* of the aspect. Initial setup takes a lot of time and specialized knowledge, and this has discouraged the use of such mechanisms except at the largest and most complex sites. While this setup is feasible, with some effort, in the networks of today, it cannot scale easily to future networks involving millions of pervasive nodes.

Practical Aspect-Oriented Design

So what does this mean to the practical administrator? Aspects are a way of *thinking* about the configuration management problem. They are a planning tool for distributed characteristics. When aspects overlap or work in concert, coordination is necessary to avoid contradictions. But, looking deeper, there are immediate benefits to thinking about and designing systems in terms of aspects, rather than basing design upon the capabilities of existing tools. In fact, it seems that the most effective way to save money spent on configuration management is not to utilize powerful tools, but to instead *refine the problem description* so that management difficulties are reduced or even eliminated.⁶

For example, it is common practice to run one kind of service per server-host, where possible. Why? In our present model, we understand this: running more than one service can lead to overlapping of aspects, because certain parameters might be needed by more than one service, risking the possibility of contradictions, and making the problem of maintaining and updating the server potentially more complex. Virtualization now provides a low-cost method for implementing the one-host-to-one-service practice, by simulating several independent servers with one physical machine.

Note that there are situations in which very complex systems exhibit no costly overlaps, again as a result of careful analysis. Consider, for instance, a linux workstation image consisting of a pre-tested suite of applications, managed as a unit, such as Red-Hat Enterprise Edition. We might want to think of the resulting workstation as a product of complex dependencies, but we can pay to have others do that thinking for us and manage each workstation as a unit. Thus we reduce cost by outsourcing the management. This is fine provided that we do not construct our own aspects that interfere with that remote management.⁷

⁶MB: Ideally these refinements would be equivalent, but our failure to model CM adequately in the past has led to a gap between common sense and technology.

⁷AC: Herein lies an important observation: *system administrators often construct aspect overlaps in crafting the requirements for a system.*

Rules for efficient aspect-oriented design of networks are thus simple and straightforward and are easily motivated by promise theory:

1. Factor services onto independent closures, e.g., virtual machines where possible (to eliminate aspect overlaps).
2. One manager for one aspect. Maintain clear separations in the source of aspect control, i.e., avoid interfering with systems that manage their own aspects, e.g., RPM and RedHat Enterprise.
3. Specify replicated parameters at a single source, e.g., one configuration file.
4. Document all remaining overlaps, so that future administrators will not fall into the trap of violating their constraints.

These rules seem intuitive, indeed they arise naturally in Service-Oriented Architectures (SOAs), which we come to shortly.

Simplest is Best?

How do we know when we have made configuration management as simple and straightforward as possible? Simple is relative, but many will agree that simple as possible is when there are minimal aspect overlaps, and the aspects in force are as *unrestrictive* as possible. Conversely, a site is “complex to manage” when overlaps cannot be eliminated and severely restrict choices. If there were an automated and reliable way to enforce an aspect via a software tool, then the complexity of managing that aspect is the complexity of managing the interface to the aspect, not the aspect itself.

At this point, we digress briefly to comment upon the relationship between the distributed aspect management problem and SOAs. This discussion will lay the groundwork for discussing methods of implementing the distributed constraints we have been discussing, and the next concept: closures.

Service-Oriented Architectures

Service Oriented Architectures (SOA) are currently in vogue. They ascend along with a heightened interest in outsourcing and delegation of responsibility in commerce. Clients need to be able to buy and sell services without surrendering their autonomy, or right to decide. SOAs enable the construction of distributed computing applications from a collection of autonomous, cooperating services. One does not expect that all the parts of the system are under the same jurisdiction.

For example, we no longer think of a “web application” as living on a single “web server”; the application is instead composed from the interactions of autonomous components, and linked via middleware that utilizes the Simple Object Access Protocol (SOAP) to expedite requests. The application thus spans several physical machines and perhaps even several enterprises, utilizing components from each.

No single administrator controls the configuration of this arrangement.

Learning from SOAs

Let us be clear: we are not advocating the use of web services for configuration management. There are many reasons why this would not be the best solution. However, service architectures embody some compelling ideas that we can utilize in configuration management:

1. Subscribing to a service is not a simple matter of pointing each client at a server. It involves some form of service guarantee as well.
2. The protocols by which one receives a service are defined by the servers of the service, using a transaction that defines required inputs and their formats.

A compelling feature of SOAs is that the process of binding client to server is not just a matter of pointing each client at a server, but involves a two-sided agreement to provide and to utilize services. This means that in an SOA, one *manages service bindings* rather than *managing service references*. With some careful thought, we can apply this practice to non-web services such as DNS, DHCP, and the like. This is a key idea that we will develop further throughout the paper.

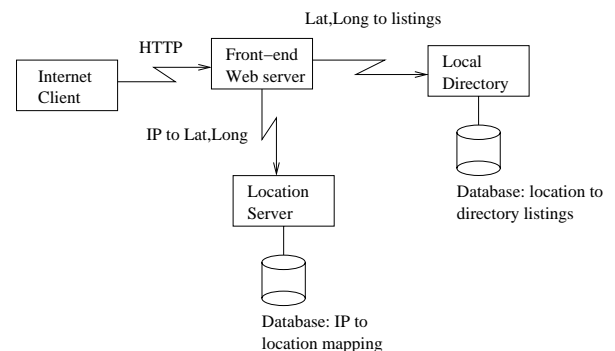


Figure 1: A service-oriented architecture in which a web application is composed from remote location and directory services.

As an example of a service-oriented application, consider Figure 1. There, a web application is created from a front-end server, bound to both a location service that maps IP addresses to latitude and longitude, and a geographically-aware search service that returns results for matching businesses, indexed by latitude and longitude.

This example exhibits many properties of an SOA:

1. Services are *autonomous* and even managed by different corporate entities.
2. Services are coordinated via negotiation between client and server.
3. There is a concept of quality of service that defines how quickly a server should respond to a request.

4. Clients can – at their option – change service providers dynamically, based upon whether a current provider is functional or not.

SOA and Configuration Management

A large part of configuration management involves receiving and utilizing remote services. However, theoretical results for configuration management have primarily concerned the low level practice of controlling bits on local disk or processes on a local system.

By contrast, the service concept deals with the intermediary issues of creating a distributed application. If we can view a computer network as a whole, as if it were an SOA application, then the problem of configuration management becomes primarily a binding problem between services and clients. This binding problem involves both network client transactions (entity to entity or peer to peer) and local transactions on the client machine (to make the machine able and ready to receive services).

There are thus two ways in which the service model applies to configuration management:

- As an information binding between hosts or peers in a network.
- As an information binding between data objects internal to a local host.

The components of an SOA application are known commonly by another name: *closures* [15, 16]. A closure is nothing more than an encapsulation of a service, that is – to some extent – self-managing. While closures do not need to comply with service standards such as SOAP or WSDL or WSAPI, any service that complies with these standards is in fact a closure. It is not surprising, therefore, that closures and SOA applications have some of the same strengths and limitations.

Closures

A closure is a domain of “semantic predictability” in which inputs result in outputs with a predictable structure. The central property of a closure is that of freedom from unknown effects; its behavior is completely determined by its transactions with the outside world, defined as input that it receives from various sources.

The configuration of a closure can be thought of as the sum of its transactions with the outside world, so that each output from a closure – in terms of behavior – is a function of all input received so far. Input can take many forms, including transactions, events, streams, etc. The only hard requirement for a closure’s input is that it must be equivalent to a serializable source, i.e., one must be able to express “what happened” as a series of occurrences, including inputs, events, etc.

Definition 10: A closure D is a service $\langle C_D, F_D \rangle$ where C_D describes constraints on input and F_D

is a function mapping inputs to responses. F_D maps each sequence S of input transactions, each of which obeys constraints C_D , to a unique output $F_D(S)$ (which may be empty).

This is a strange definition that is difficult to appreciate until one looks at its opposite. A closure is like a service whose output is a *function* of the totality of its input. The alternative is a service whose output is not such a function, i.e., its output varies with respect to other sources than just what you tell it. The crucial property that determines whether we have a closure or not is “complete knowledge” of all operations that might change its output. Any system in which we can claim such knowledge is said to be “closed,” while a system in which we cannot make the claim remains “open,” i.e., closures are “fully determined.”⁸

Example 8 The simplest possible closure is one that memorizes a mapping, e.g., a simple version of DNS. Inputs to the closure include queries that inquire about mappings, as well as transactions that change mappings. While no query changes a mapping, transactions do. So the result of a query is always the result of the sum total of the prior transactions that specify mappings. Since these transactions take the form of reloading the configuration file and DNS by nature forgets all but the last such transaction, the result of a query is completely determined by the last transaction of reading DNS configuration. This suffices to make DNS a closure. DDNS is also a closure, provided that we count DDNS assertions as transactional inputs.

Example 9 A database server is a closure; the result of a query depends upon all prior commands given to the server, all the way back to “create database.”

Example 10 A “business data object” in a service-oriented architecture is a closure; it defines transactions (SELECT, INSERT, MODIFY, DELETE) that can change data state and presumes that no other operations will be utilized.

A thing is not a closure if there is a way that the service response can change without a transaction, or not as a function of transactions.

Example 11 If human administrators manually make changes to a system that expects to be manipulated only by a strict transaction protocol, closure will be broken.

⁸MB: Closures are a computer science idealization to my mind. They ignore the effect of outside influences that one cannot necessarily control, e.g., mistakes made by inexperienced prying hands, i.e., they conjecture that we have more control over a system than is realistic. But they are still useful ways of talking about operations, that can be made approximately correct provided we make sure they are maintained using additional constraints such as iterative, convergent maintenance [7].

The way this usually occurs is for something that can change the output to remain unknown to the person interacting with the system. If you define transactions as “actions taken by one system administrator” and – unknown to you – there is *another* system administrator clandestinely configuring the system, you do not have a closure. Thus, rather trivially,

Theorem 1 *Any closure’s behavior can be emulated by a set of SQL transactions, in which each closure transaction is translated into an SQL equivalent.*

Proof 4 *First, consider each closure transaction as a command with parameters. Translate those parameters into SQL parameters. The definition of a closure is that its output is a function of its input, where some transactions may be ignored. As SQL is Turing-Universal, the intent of each transaction can be translated into SQL, using the underlying database as the “Turing tape.” Thus the behavior of a closure can be emulated by SQL, as it can be emulated in any other programming language.*

In defining a closure, we have intentionally put as little structure into the closure as possible. Structure is imposed by algebraic rules that simplify the book-keeping we must do to compute a closure output. These rules tell us when an input is *not* operative in producing an output, and define equivalence classes for input streams that produce the exact same output.

Example 12 *Suppose we have a simple closure that does nothing but store and retrieve string parameter values. It has two input events, GET and SET, where the last SET determines the value of the next GET for a parameter. This last sentence says it all: GETs do nothing to modify state; SETs do modify state. Thus the next value for a GET is determined by the sequence of last SETs for each parameter, and the order of these SETs is not important once we have deleted previous SETs of the same parameter. Thus the simplicity of this closure results from the algebraic property that all transactions are stateless [17].*

The power of closures arises not from the relatively awkward definition, but from the fact that many common closures are easy to describe *algebraically*, in similar fashion to the examples above. Let us consider the algebraic properties of a selected group of closures.

Example 13 *We can think of a DNS server as receiving transactions about mappings from around the world, and queries that depend upon those mappings. At any one time, the result of a query is the last received mapping.*

Example 14 *We can think of a file-server as receiving (block-level) transactions to write blocks and returning (block-level) reads. At any*

time, the result of a read has the content written during the last write of that block.

Example 15 *A web service closure [15] has inputs consisting of queries and mappings. Queries do not affect mappings, while mappings directly affect which page is returned for a query.*

Several other properties of closures are worth repeating from [15]:

1. Closures are a unit of *independence* in a configuration; the closure only behaves according to the inputs it receives, and no others.
2. Closures can span network nodes and constitute the behavior of peer-peer infrastructures, e.g., DNS.
3. Closures can communicate amongst themselves to create larger closures, e.g., combining web, DNS, DHCP, and routing layers.

We need closures to understand aspect implementation, so let us look at how the two relate.

Closures and Aspects

The main difference between closures and aspects is the use of interior versus exterior constraints. A closure’s constraint model is *exterior*; its behavior is defined as a function of its inputs, with no reference to *how* that behavior is assured. An aspect has no explicit concept of behavior; it is instead an interior measure of how something should be configured with implicit consequences; the behavior of an aspect is exterior to its definition. In other words, an aspect is a declarative concept with implicit behavioral consequences.

The mapping between configuration and behavior has been systematically studied in [18] and we adopt the notation of this work here. Behavior is abstractly represented as a subset of a set of tests that can be either true or false. We can think of the current state of a system as a “subset of known symptoms” that can be observed. The subset consists of the tests that are true under a given condition. The behavior of an aspect is a relation describing the correspondence between aspect values V and aspect behaviors T : a set of ordered pairs (V, T) where V is an aspect value and T is a subset of tests from a test suite \mathcal{T} that are true when the aspect has the corresponding value. This may be a function of the aspect’s value, or instead a relation between an aspect and many possible sets of test outcomes.

Definition 11: *An aspect is closed with respect to a specific set of behaviors \mathcal{T} if there is a map between values for the aspect and behaviors that are exhibited. If not, the aspect is open with respect to \mathcal{T} .*

Note that this definition is carefully crafted. There is no such thing as an aspect closed with respect to every behavior; one must select a set of behaviors to observe. Likewise, it is usually possible to find some behaviors that are uniquely determined by aspect

choices; these are the *closure behaviors* of an aspect (or even the *closure of the aspect*).

Lemma 2 *The closure behaviors of an aspect, together with the aspect, form a closure.*

Proof 5 *The aspect – as a potential closure – has inputs that are the values of configuration parameters and outputs that are sets of behaviors that may or may not be exhibited by those choices. By definition of aspect closure behaviors, these behaviors do not change except as a result of parameter changes, which may or may not be accomplished by other interactions. Considering the stream of inputs, including parameter changes, the definition of a closure is met.*

In other words, *any aspect is a closure with respect to a selected set of behaviors.*

This fact has important implications. The boundaries of closures are determined by what is expected, and what is known, not by what is controlled. Aspects give us a way of easily constructing closures, which we did not have before. The key to constructing a closure is to *think* about an aspect in the proper way, and define “enough” to close it! In other words, *closures are not so much constructed as much as they are discovered.*

Practical Closures

Closures – like aspects – allow one to *think* about the configuration management problem efficiently and effectively. Recall that the complexity of aspect composition depends upon the amount of overlap, and the overlap is determined by the *interface* to the aspect in question. Then add the concept that an aspect is closed if its behavior over a parameter set is predictable.

Principle 1 *The manageability of an aspect, relative to a fixed set of tools, is increased by limiting variability of parameter values for the aspect.*

In other words, if one can “get away” with a small number of variations, then the aspect becomes easier to think about and manage. And, if one limits sufficiently and circumscribes its behavior accurately enough, it becomes a closure. For example, we might limit the way hosts bind to databases: innocent at first glance, but a good cost-saving mechanism.

In previous closure work, the implication was that closures are *constructed* by building complex interface code. This simple analysis shows that closures are instead *discovered* by factoring otherwise complex systems.

Promises

No general language has been developed to describe how aspects can be managed or described at a low level, nor how closures should communicate in order to implement reliable management changes. So far, the language of closure communications has been in terms of “demands” and “acknowledgments.” However, as we have already commented, this idea of

making demands is intrinsically at odds with the reality of distributed systems.

The fact that different decision agencies are involved in distributed systems changes many things. One can no longer imagine being in complete control of a network of hosts, making demands, unless everyone agrees to behave in a subordinate fashion and comply with our expectations of them. This brings us to the notion of *promises* and, coincidentally, back to something like a Service Oriented Architecture.

Voluntary Cooperation

If we cannot guarantee behavior by requirement or demand, can we at least make agreements with components of the system to behave in a manner that is acceptable to us? Service Level Agreements (SLA) are one manifestation of this realization for commodity services. These are familiar to most of us. However, Service Level Agreements are too vague and too complex a construction to be useful for analysis. We therefore introduce the atomic idea of a *promise* [10, 12, 14].

We begin with the players in the system that make promises.

Definition 12: Agents *An agent is any entity within any system that can make or receive promises, and which computes all decisions autonomously. The information within an agent is not available to any other agent, unless that availability (of information) is promised.*

An autonomous agent cannot be forced or coerced into any behavior against its will.⁹ In particular, one cannot demand or require anything of an autonomous agent, one can only suggest or request something of it, by expressing a willingness to receive and use a service that one hopes it will promise to provide.

Because we are developing a language for abstracting cooperative agreements, we are free to apply this model to a variety of scenarios, even where agents represent “dumb” resources like disks or files if we choose. This does not mean that files are intelligent, it simply means that someone is controlling the file’s properties and behavior as an independent object, and this abstraction allows us to describe that interaction in low level atomic terms.

Definition 13: Promise *A promise is a specification of future state or behavior from one autonomous agent to another. It is thus a unit of policy. A promise is a link in a labeled graph $G = \langle A, L, \Pi \rangle$ in which the set of nodes A are agents, the directed edges or links L are promises and the labels Π are called the promise body. A promise is a private announcement $\pi \in \Pi$ from the sender node s of the promise to the receiver r . We denote it like this:*

$$s \xrightarrow{\pi} r \quad (1)$$

meaning that s promises π to r .

⁹MB: This property usefully agrees with the security model used by cfengine.

This definition agrees with our commonplace understanding of a promise, but is sufficiently formal that we can use it for analysis. Promises will form a building block for aspect closures, and will allow us to rewrite familiar concepts of configuration management such as *operators* [8] entirely in terms of voluntary cooperation.

Give and Take

A promise is a specification of behavior, but it might be unclear at this point how a promise might describe behavior. There are several principles that apply to this description:

1. A promising agent can only describe its own behavior or behaviors of others that it has directly observed.
2. That behavior includes the properties of a specific set of *interactions* that may occur with neighboring agents.
3. There is no reason to identify an autonomous agent with a “system” or a “machine.” One can create many autonomous agents within a single system, as components.

There are two primitive types of promise body from which it is believed all others can be constructed: these are the service and acceptance promises (or promises to given and take). In addition it is convenient to define a third type called the coordination promise [14] as a shorthand.

- A service (giving) promise, whose body is denoted π , is the basic type of promise which denotes a restriction of behavior by the promising agent in the manner of a service:

$$a \xrightarrow{\pi} b \quad (2)$$

involves an offer of service from a to b and implies a specification of future behavior of a towards b .

- A usage or acceptance promise (taking), denoted $U(\pi)$, is the promise to receive or use a service π promised by another agent.

$$a \xrightarrow{U(\pi)} b \quad (3)$$

involves a receipt of information and service by the promising agent a from b . It can be related to access control, for instance.

- A coordination (or subordination) promise, denoted $C(\pi)$, is the promise to do the same as another agent with respect to a promise body π .

$$a \xrightarrow{C(\pi)} b \quad (4)$$

involves that b informs a about its actions with respect to promises of type π , and the receipt and usage of that information by a . This promise is a subordination because a is willingly giving up its autonomy in the matter of π by agreeing to follow b 's lead. Note that this agreement is made on a peer to peer basis, and implies no *a priori* centralization.

Types of Promise

Promises are only useful if they can be made about many kinds of issues. To distinguish between kinds of promises, each promise body consists of two

parts: a *type* τ which labels the issue being addressed along with its possible domain of variability, and a *constraint* C which tells us which subset of the domain of possibility for that type is being promised.¹⁰

Example 16 Consider a configuration promise. Suppose that τ represents a configuration parameter belonging to the promiser and $C \subseteq \tau$ represents a set of allowable values for that parameter that are allowed by policy. Then $pi = \langle \tau, C \rangle$ is a promise that the values C will be adhered to as a value for the parameter described by τ .

Note that a configuration *parameter* is a syntactic thing, while a *promise* about that parameter constitutes a form of *knowledge*. It is best to think about active promises as a form of “distributed knowledge” about a system. When an entity promises something, it limits its behavior in observable ways. The union of “promises made” is a form of distributed system state.

Example 17 Suppose that τ represents a subset of parameters, belonging to a single host, within a distributed aspect and C represents some constraints on those parameters. Then the promise represents a policy atom on the particular promising host that expresses its personal part of that aspect. In other words, a promise expresses limits imposed upon an aspect by one individual agent.

Promises can be combined into knowledge about the network. The method of combination of promise information is specific to the kind of promise.

Example 18 Suppose that we take the point of view of a single autonomous agent A_0 . Agent A_1 promises agent A_0 that it is a directory server (constraint C_1), and agent A_2 promises A_0 the same thing for itself (constraint C_2). Then the two promises offer alternatives to the receiver but do not oblige it in any way. The result is that the receiver is free to assume the logical-or of the input promises ($C_1 \vee C_2$).

Example 19 Suppose we again take the point of view of a single autonomous agent A_0 . Suppose agent A_1 promises to A_0 some information (i.e., it is constrained to provide that information) and that the information is part of a distributed aspect, e.g., it informs A_0 where to find DNS service. This information in no way obliges A_0 to use that information. However, if A_0 promises A_1 to use that information, it is constrained to follow A_0 's suggestion.

The important point from these examples is that what an agent does with promises, and the meaning of combining them, is entirely up to the individual agents. Autonomous agents, like closures, have the property of being capable of engaging in arbitrary reasoning based upon the inputs they are given.

¹⁰AC: It seems that no matter how flexible I am about interpreting this definition, a promise is more general than that! “But wait, there’s more!”

Promise Notation

Promise graphs become complex quickly and are difficult to notate other than in pictures. To ease notation, we adopt some simple notational conventions. First, if a set of nodes is involved in making the same promise, we unambiguously represent the set of promises as a single promise between sets. If S and R are sets, then $S \xrightarrow{\pi} R$ means the set of promises $s \xrightarrow{\pi} r$, for $s \in S$ and $r \in R$. Similarly, $S \xrightarrow{\pi} r$ is the set of promises $s \xrightarrow{\pi} r$, for $s \in S$, and $s \xrightarrow{\pi} R$ represents the set of promises $s \xrightarrow{\pi} r$ for $r \in R$.

It is also often interesting to know which kinds of promises have been made, without knowing necessarily who made them or to whom they were made. We write $S \xrightarrow{\pi} (.)$ to mean that each $s \in S$ has made the promise π to some unknown set of hosts, and $(.) \xrightarrow{\pi} R$ to mean that some node has promised π to each $r \in R$.

Roles

An important concept in promises is that of a role. A role is a kind of emergent pattern that we can identify in the promises made or received by agents.

Definition 14: Role Suppose S and R are sets of autonomous agents and there is a promise of type τ between each node in $s \in S$ and each node in $r \in R$. Then S and R are said to form role-sets of type τ . S is said to have a sender role of type τ while R is said to have a receiver role of type τ .

Example 20 Let S be the set of web servers and R be the set of web clients that can access the servers S . Then $S \xrightarrow{\text{web}} R$ describes two roles: $S \xrightarrow{\text{web}} (.)$ (S are “web servers”) and that $(.) \xrightarrow{\text{web}} R$ (R are “web clients”).

If a client receives an offer of service but does not promise to use that service, the role of the client is limited to that of being promised the service. The client would have to formally agree to use the service in order to be classified as a service client according to the role model (since this implies a binding commitment).

Example 21 The simplest example of a role is that of a file server that serves home directories. The file server promises to serve up home directories to clients, and the clients in turn utilize that service in order to allow users interactive access to their files. The fact that the file server’s promise is implicit, i.e., determined by use rather than by an explicit communication, is not important. The role of “file server” is an emergent property of how the server is used, not a matter of intent.

In promise theory, one can do many things with roles. They can be composed to form composite roles (i.e., through the holding or use of more than one promise):

Definition 15: Composition of roles Suppose R_1 and R_2 are role-sets with respect to types τ_1 and

τ_2 , where the direction of each τ_i may vary. Then $\tau_1 \cap \tau_2$ is also a role.

Example 22 Suppose that a web server s sends a promise $s \xrightarrow{\text{web}} R$ to a set of clients R . Those clients who received the promise form one role R . Those clients who also responded with a promise to use form another role $R' \subset R$. The clients who for some strange reason respond with a promise to use without a matching promise to serve form a third role R'' disjoint from R . There is no particular reason that an agent cannot promise to use a service that does not exist. The distinction between these roles is whether one or two promises were made.

In the strictest interpretation of promise theory, roles are distributed aspects and therefore cannot be forced or decided by anyone. However, in practice roles can be identified empirically (a posteriori), or be decided as a design decision in advance (a priori) if we are in the fortunate circumstance of controlling several (formally) independent agents.

Lemma 3: Agents are trivially roles Let A be the set of agents. A is a role.

Proof 6 Consider the empty set of promises \emptyset . Every agent in the graph sends and receives this set of ‘no promises’ in addition to any other promises it might send or receive, thus the pattern of no promises is identified as a subset within the promise graph at every node. Hence every agent node plays a role of ‘no promise,’ which we can re-name ‘autonomous agent.’

Promise theory is essentially a model for the planning and analysis of generalized services. The challenge is to use promises to see how configuration management, perceived as a service, can be carried out by autonomous agents. We refer readers to [10, 12, 11] for more information about promises.

Promises and Closures

The relationship between promises and closures is subtle but straightforward. In all that has been published about closures, little has been said about the language utilized by closures to communicate with one another. The concept of autonomous agent, utilized in promise theory, is roughly the same as the concept of closure, though closure is more restrictive as a concept. Promises, as an inter-agent language, are an ideal mechanism with which closures can communicate.

Theorem 2 Closures are a subclass of autonomous agents.

Proof 7 Closures require that all transactions are functions of prior transactions and nothing else. This is more restrictive than the definition of an autonomous agent, which requires actions based upon autonomy and previous history, but does not limit the sources of information utilized for such actions.

Transactions include such things as making promises, but this is more restrictive than the restriction of autonomy. An agent's output could change via other mechanisms than promises or transactions, e.g., resource requirements. Also, while agents act asynchronously and without any notion of transaction, closures rely upon transactions and transaction serialization to arrive at a notion of internal state (in terms of the sequence of past transactions). So in general, a closure is an agent, but not all agents are closures. In like manner, promises are appropriate closure interactions, but not all closure interactions are promises; some are transactions in the traditional sense of being tightly coupled and not subject to debate or choice.

Client cooperation is an important way of building distributed services. On the one hand, we would like to 'demand' the compliance of services around the network, since we are used to "control" rather than "cooperation," but we cannot.

Example 23 Consider the case of a client binding to a DNS server (see Figure 2). The client can ask a candidate server for a "promise" of service. If the DNS candidate responds with an acknowledgment, this means that its side of the distributed aspect called DNS is ready to converge to a coherent and functional state. Then, when the client adds the DNS server to its resolver table, the distributed aspect becomes complete and functional.

In the figure, notice that the client makes no promises to the server. They have no agreement. Rather, the master server promises to use and requests the client sends, and to reply to them if they arrive. The relationship between master and slave is more complex. Slave status is acquired by the slave agent subordinating itself with a $C(DNS)$ promise. This means it will make the same promises about DNS that the master will. It agrees to use the zone data sent by the master. The client promises a policy adjudicator that it will contact the master server, and if there is a timeout, it will contact the slave.

Promises and Aspects

We have seen in a previous section how to view the values of aspects in a network as synonymous with its configuration. We now study service binding aspects in more detail. We show, particularly, that there is no way to separate the function of a service binding from the guarantees of function that a server can provide and, in turn, the promises the server can keep. In this way, a "promise kept" is stronger than any current mechanism for centralized control of service bindings.

First, we need a mechanism for "semantic grounding" of the promises on each host. Let Ω be an oracle that describes host documentation and the reasonable constraints of single-host configuration. Ω is

the union of all local aspects, and could be described as the source of a union of individual overlapping promises $\langle \tau, C_\tau \rangle$. In other words, constraints arising from the documentation of a system are promises of the form $h \xrightarrow{U(data)} \Omega$, where h is the local host (i.e., the host promises to comply with documentation).

Proposition 5: *The class of promises that the grounding agent makes are a role that determine the kind of machine being configured.*

Proof 8 *All machines with the same kind of architecture have the same grounded promises, hence they are members of a role by definition.*

The documentation Ω is nothing more than an embodiment or symbol of the constraints arising from the system itself.

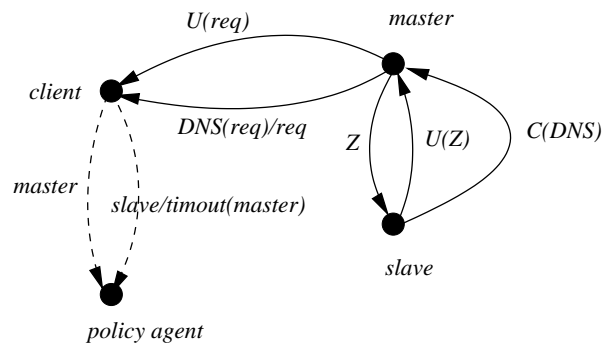


Figure 2: A promise graph for a DNS lookup.

Second, we consider the local policy on a machine as having a different form of grounding.

Definition 16: *Let Γ represent a declaration of local policy, which may change. Promises of the form $h \xrightarrow{U(data)} \Gamma$ determine desirable behavior on the local host.*

The relationship of Ω to Γ is that of a hard aspect to a soft one; limits versus desires.

In formal promise theory, Ω and Γ are possibly hidden parts of the agent; here we make them explicit only to describe the relationship between aspects and promises. Ω represents all of the hard aspects, i.e., the things about the system that are not negotiable; Γ represents soft aspects, determined by policy. These aspects are inputs to the agent's view of the world, not as binding obligations, but instead as information that the agent can use, along with all other promise information with which it is provided. Documentation and experience are as much promises as are messages from an external agent; they are guarantees of specific behavior for the underlying systems.

The point of this discussion is that for all practical purposes, everything an agent needs to do or know about the world can be expressed by some kind of promise. Some of these promises come from other agents.

We emphasize once again that agents in promise theory are not to be confused with configuration agents

(e.g., cfagent), their closest interpretation would be individual configuration objects such as files or processes.

Distributed Aspects

A distributed aspect has often been viewed as “pointing” clients to a specific server. We take a different view, in which both client and server take responsibility in a more fundamental way. The key to our argument is the following simple idea:

Proposition 6: *A binding is a transaction between service provider and service consumer, in which the server guarantees to reliably provide a service while a client guarantees that it will also reliably consume the service.*

This simple idea has such subtle ramifications that it must be studied in some detail to understand the text that follows.

Example 24 *In a typical configuration management scenario, a binding is a simple concept of “naming a server” in some context. We “point” our resolver at a “DNS server,” or “point” our outgoing mail at a “mail relay.” This “pointing” is a matter of blind faith; we assume at some level that the servers we are pointing to are actually providing the service we require, and that some mechanism, either ours or someone else’s, has configured them properly to provide that service.*

We wish to challenge this idea of binding in an extremely straightforward (and even seemingly trivial) way.

Principle 2 *A distributed aspect (e.g., a client-server binding) is configured correctly only if both sides of the client-server relationship are both conversing with the appropriate server and functioning properly as server and client.*

This may seem silly as a principle. Everyone knows this, except that we usually configure the server and client *separately* and manage the two entities as *separate aspects*. It is unfortunate that we also tend to think of these aspects as separate entities as well. In a sense, we do not acknowledge the *distributed aspect* that consists of both of these functioning together, correctly. It is this aspect, not the individual servers, that we are responsible for managing. In other words, a promise is more than a pointer. It is a “guarantee,” somewhat like that contained in an SLA, that a service is up and running and answering queries.

This simple way of thinking leads to a drastically different understanding of configuration management as a practice. The “master-slave” view of configuration management is that we have to make all the servers work correctly, and point clients at servers (taking it on faith that the servers will function properly when pointed to), and everything will just work. The reality is that each binding between a client and a server is something that must work properly as a *distributed aspect*. This may require some coordination

between server and client that we tend to ignore, but that is crucial to network function.

Example 25 *A very simple example of a distributed aspect with non-trivial behavior arises from incompatibilities between server and client parts of NFSV3 when utilized over a specific router between a Sun file-server and a Linux client. The aspect, to function properly, must utilize NFSV2 instead. The reason that this is true is not a function of either the server or the client, but of the router between them! Correct function of the client and server is not relevant to function of the aspect; a third piece of the puzzle constrains behavior further and – without that piece – two perfectly configured hosts fail to interoperate. Most important, this behavior of the binding remains invisible unless one looks at the behavior of the whole binding, rather than the behavior of its endpoints.*

Using Promises

A next-generation configuration management tool might utilize promises in an extremely straightforward way. A configuration tool (let us avoid the confusion of calling it an agent) runs on each host to manage service bindings. The tool running on a host providing a service declares this fact via a number of promises. All service bindings are based upon promises received. Among promises received, arbitrary choices are made as to which servers to use, or perhaps some primitive form of distance calculation is utilized to determine the “nearest” server from among several candidates.

It is important to note that every promise received corresponds to a *functional* machine providing a service, not just a pointer to a machine that may or may not be working at the time. So the problem of pointing machines to non-existent services disappears. Every use-promise informs a server about which agents to contact if an outage is expected. This gives the clients time to re-organize their bindings to point to usable servers during the outage.

Example 26 *Consider the often costly problem of maintaining default printers for desktop workstations and remote users. We want the default printer to be “near” to the user or desktop, presenting an ongoing and expensive management problem as printers and desktops are installed or retired. Now consider the same binding problem and apply promise theory, running an agent to report upon the status of each printer, and bind those agents into a role. The centralized database of nearest printers is replaced by a series of local databases, one for each agent, defining the nearest desktops to their printers. Maintaining this information requires only notifying a single local agent of a change, rather than the whole database of nearest printers. Determinism is*

preserved without centralization, and the management problem is naturally distributed to agents within the control of each separate administrative domain. Yes, as the reader might be guessing already, this is a closure as well!

Example 27 Consider the problem of maintaining resolver bindings in the presence of network changes, and apply the same architecture of distributed agents as above. Each DNS server reports its availability to all consuming agents, and they can bind at will. This enables online load-balancing using caching and stealth servers, without reconfiguring the network for each addition or deletion of server.

Example 28 Consider the problem of determining primary gateways for each host. This is already solved through routing protocols which, if one considers them carefully, consist entirely of promises.

Shedding Light on Configuration Management

A *taxonomy* is a description of the space of options for a thing. In this case, the “thing” in question is the practice of configuration management. Using aspects, closures, and promises, one can describe many current configuration management strategies, and compare them within that theoretical framework. This gives us a fundamental idea of each strategy’s strengths and limits.

A typical user of CFEngine is using promise theory without knowing it. The cfagent process receives a configuration file from a central server that – in its essence – contains lots of promises. Instructions that bind the host to specific servers can be interpreted as promises (from the master server) that the services will be present and available. The exact same file enforces distributed aspects, and may in fact determine closures, via its contents and ability to correct errors. The complexity of this file is its main weakness; promises, aspects, and closures offer a way to conceptually simplify its contents in the future.

A typical user of configuration scripts uses promises in a much simpler way. The user of a script is – in essence – personally promising that the script will work, which in turn is the same thing as promising that the configuration settings changed by the script are appropriate and will have appropriate effects. Again, the concepts of aspects and closures are implicit and well-hidden within the script; we cannot currently analyze scripts in enough detail to infer the reasons for a change from the script that makes the change.

A typical user of LCFG, BCFG2, or other generative tools depends upon the tool to hide information about promises, aspects, and closures that the tool creates and manages. The strength of these tools is information hiding; the user need not cope with the true complexity of aspects. But at the same time, the

centralized planning functions of these tools cannot react automatically to distributed changes (e.g., between autonomously managed domains) so that promises may provide a way to make these tools more adaptive to changes in network state.

Conclusions

We have seen in this paper how the concepts of closures and promises – seemingly very different – are actually sides of the same coin. The “glue” by which this comparison is made is the concept of an “aspect,” as well as the idea that a configuration is a composition of overlapping aspects. Aspects are important because they are closer to the way in which administrators currently think. As Paul Anderson has noted on several occasions, the challenge for the future is to look for ways to compile high level aspects into low level operations. We believe that this goal is now much clearer from our formalizations. We now have a complete story that captures and unifies all of our state of the art understanding of configuration management:

1. Aspects are constellations of promises.
2. Promises with their agents can form closures.

We identify a progression from high level to low level:

High level → Low level

Planning → Implementation

Aspects → Promises

This progression makes no assumptions about centralization or authority, nor does it have to be a linear progression. One can approach it “top-down” or “bottom-up” [8], as one sees fit. Not every aspect is necessarily implementable, if the associated promises are not made (or kept), and we can discover this by attempting the decomposition from high level goals to low level implementation.

It follows from the requirement of convergence that observation is a key element in configuration management [9, 18]. The separation of change management from monitoring is a fundamental mistake in current systems. These issues need to be tightly woven to make reliable bindings with predictable service agreements. It is our belief that a next generation of configuration management tools can do this, utilizing promises, aspects, and closures as conceptual parts of designing and architecting an efficient and robust configuration management strategy.

Acknowledgment

We are grateful to Paul Anderson, Jan Bergstra, and Aileen Frisch for formative discussions. Narayan Desai, Marc Chiarini, Ning Wu, Hengky Susanto, Josh Danziger, and Bill Bogstad gave helpful comments on initial drafts.

Author Biographies

Mark Burgess is a Professor of Network and System Administration at Oslo University College,

Norway. He is the author of cfengine and several books and papers on system administration. He can be reached by electronic mail as Mark.Burgess@iu.hio.no.

Alva Couch is an Associate Professor of Computer Science at Tufts University. He is an author of numerous papers on the theory and practice of system administration, and currently serves as Secretary to the USENIX Board of Directors. He can be reached by electronic mail as couch@cs.tufts.edu.

Bibliography

- [1] Anderson, P., "Towards a high level machine configuration system," *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, USENIX Association, Berkeley, CA, p. 19, 1994.
- [2] Anderson, P., *System Configuration, SAGE Short Topics in System Administration*, 2006.
- [3] Anderson, P., P. Goldsack, and J. Patterson, "Smartfrog meets lcfc: Autonomous reconfiguration with central policy control," *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)*, USENIX Association, Berkeley, CA, p. 213, 2003.
- [4] Burgess, M., "A site configuration engine," *Computing systems*, MIT Press, Cambridge MA, Vol. 8, Num. 309, 1995.
- [5] Burgess, M., "Automated system administration with feedback regulation," *Software practice and experience*, Vol. 28, p. 1519, 1998.
- [6] Burgess, M., "CFEngine as a component of computer immune-systems," *Proceedings of the Norwegian conference on Informatics*, 1998.
- [7] Burgess, M., "On the theory of system administration," *Science of Computer Programming*, Vol. 49, p. 1, 2003.
- [8] Burgess, M., *Analytical Network and System Administration – Managing Human-Computer Systems*, J. Wiley & Sons, Chichester, 2004.
- [9] Burgess, M., "Configurable immunity for evolving human-computer systems," *Science of Computer Programming*, Vol. 51, p. 197, 2004.
- [10] Burgess, M. and S. Fagernes, "Pervasive computing management: A model of network policy with local autonomy," *IEEE Transactions on Software Engineering*, (submitted).
- [11] Burgess, M. and S. Fagernes, "Pervasive computing management: Applied promise theory," (preprint), (submitted).
- [12] Burgess, M. and S. Fagernes, "Pervasive computing management: Policy through voluntary cooperation," (preprint), (submitted).
- [13] Burgess, M. and R. Ralston, "Distributed resource administration using cfengine," *Software practice and experience*, Vol. 27, p. 1083, 1997.
- [14] Burgess, Mark, "An approach to understanding policy based on autonomy and voluntary cooperation," *IFIP/IEEE 16th international workshop on distributed systems operations and management (DSOM)*, in LNCS 3775, pp. 97-108, 2005.
- [15] Couch, A., J. Hart, E.G. Idhaw, and D. Kallas, "Seeking closure in an open world: A behavioural agent approach to configuration management," *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)*, USENIX Association, Berkeley, CA, p. 129, 2003.
- [16] Couch, A. and S. Schwartzberg, "Experience in implementing an http service closure," *Proceedings of the Eighteenth Systems Administration Conference (LISA XVIII)*, USENIX Association, Berkeley, CA, p. 213, 2004.
- [17] Couch, A. and Y. Sun, "On the algebraic structure of convergence," *LNCS, Proceedings 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Heidelberg, Germany, pp. 28-40, 2003.
- [18] Couch, A. and Y. Sun, "On observed reproducibility in network configuration management," *Science of Computer Programming*, Vol. 53, pp. 215-253, 2004.
- [19] Couch, A. L., N. Wu, and H. Susanto, "Towards a cost model for system administration," *Proceedings of the Nineteenth Systems Administration Conference (LISA XIX)*, USENIX Association, Berkeley, CA, pp. 125-141, 2005.
- [20] Desai, N., R. Bradshaw, S. Matott, S. Bittner, S. Coghlan, R. Evard, C. Lueninghoener, T. Leggett, J.-P. Navarro, G. Rackow, C. Stacey, and T. Stacey, "A case study in configuration management tool deployment," *Proceedings of the Nineteenth Systems Administration Conference (LISA XIX)*, USENIX Association, Berkeley, CA, p. 39, 2005.
- [21] Finke, J., "Automation of site configuration management," *Proceedings of the Eleventh Systems Administration Conference (LISA XI)*, USENIX Association, Berkeley, CA, p. 155, 1997.
- [22] Finke, J., "An improved approach for generating configuration files from a database," *Proceedings of the Fourteenth Systems Administration Conference (LISA XIV)*, USENIX Association, Berkeley, CA, p. 29, 2000.
- [23] Holgate, M. and W. Partain, "The arushra project: A framework for collaborative UNIX system administration," *Proceedings of the Fifteenth Systems Administration Conference (LISA XV)*, USENIX Association, Berkeley, CA, p. 187, 2001.
- [24] Kanies, L., "Isconf: Theory, practice, and beyond," *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)*, USENIX Association, Berkeley, CA, p. 115, 2003.

- [25] Narain, Sanjai, "Network configuration management via model finding," *Proceedings of the Nineteenth Systems Administration Conference (LISA XIX)*, USENIX Association, Berkeley, CA, p. 155, 2005.
- [26] Patterson, D., "A simple way to estimate the cost of downtime," *Proceedings of the Sixteenth Systems Administration Conference (LISA XVI)*, USENIX Association, Berkeley, CA, p. 185, 2002.
- [27] Roth, M. D., "Preventing wheel reinvention: the psgconf system configuration framework," *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)* USENIX Association, Berkeley, CA, p. 205, 2003.
- [28] Sun, Y. and A. Couch, "Global impact analysis of dynamic library dependencies," *Proceedings of the Fifteenth Systems Administration Conference (LISA XV)* USENIX Association, Berkeley, CA, p. 145, 2001.
- [29] Traugott, S., "Why order matters: Turing equivalence in automated systems administration," *Proceedings of the Sixteenth Systems Administration Conference (LISA XVI)*, USENIX Association, Berkeley, CA, p. 99, 2002.
- [30] Traugott, S. and J. Huddleston, "Bootstrapping an infrastructure," *Proceedings of the Twelfth Systems Administration Conference (LISA XII)*, USENIX Association, Berkeley, CA, p. 181, 1998.
- [31] Wang, Yi-Min, Chad Verbowski, John Dunagan, Yu Chen, Helen J. Wang, Chun Yuan, and Zheng Zhang, "Strider: A black-box, state-based approach to change and configuration management and support," *Proceedings of the Seventeenth Systems Administration Conference (LISA XVII)*, USENIX Association, Berkeley, CA, p. 159, 2003.

Windows XP Kernel Crash Analysis

Archana Ganapathi, Viji Ganapathi, and David Patterson
– University of California, Berkeley

ABSTRACT

PC users have started viewing crashes as a fact of life rather than a problem. To improve operating system dependability, systems designers and programmers must analyze and understand failure data. In this paper, we analyze Windows XP kernel crash data collected from a population of volunteers who contribute to the Berkeley Open Infrastructure for Network Computing (BOINC) project. We found that OS crashes are predominantly caused by poorly-written device driver code. Users as well as product developers will benefit from understanding the crash behaviors elaborated in this paper.

Introduction

Personal Computer (PC) reliability has become a rapidly growing concern both for computer users as well as product developers. Personal computers running the Microsoft Windows operating system are often considered overly complex and difficult to manage. As modern operating systems serve as a confluence of a variety of hardware and software components, it is difficult to pinpoint unreliable components.

Such unconstrained flexibility allows complex, unanticipated, and unsafe interactions that result in an unstable environment often frustrating the user. To troubleshoot recurring problems, it is beneficial to data-mine, analyze and document every interaction for erroneous behaviors. Such failure data provides insight into how computer systems behave under varied hardware and software configurations.

To improve dependability, systems designers and programmers must understand operating system failure data. In this paper, we analyze crash data from a small number of Windows machines. We collected our data from a population of volunteers who contribute to the Berkeley Open Infrastructure for Network Computing (BOINC) project. As our analysis is based on a small amount of data (with a self-selection bias due to the nature of BOINC), we acknowledge that our results do not represent the entire PC population. Nonetheless, the data reveals several useful results for PC users as well as researchers and product developers.

Most Windows users have experienced at least one “bluescreen” during the lifetime of their machine. A sophisticated PC user will accept Windows crashes as a fact and attempt to cope with them. However, a novice user will be terrified by the implications of a crash and will continue to be preoccupied with the thought of causing severe damage to the computer. Analyzing failure data can help users gauge the dependability of various products and understand the source of their crashes.

From a research perspective, the motivation behind failure data-mining is manifold. First, it reveals

the dominant failure cause of popular computer systems. In particular, it identifies products that cause the most user frustration, thus facilitating our efforts to build stable, resilient systems. Furthermore, it enables product evaluation and development of benchmarks that rank product quality. These benchmarks can influence design prototypes for reliable systems.

Within an organization, analyzing failure data can improve quality of service. Often, corporations collect failure data to evaluate causes of downtime. In addition, they perform cost-benefit analysis to improve service availability. Some companies extend their analyses to client sites by gathering failure data at deployment locations.

For example, Microsoft Corporation collects crash data for their Windows operating system as well as applications used by their customers. Unfortunately, due to legal concerns, corporations such as Microsoft will usually not share their data with academic research groups. Companies do not wish to reveal their internal vulnerabilities, nor can they share third party products’ potential weaknesses. In addition, many companies disable the reporting feature after viewing proprietary data in the report. While abundant failure data is generated on a daily basis, very little is readily sharable with the research community.

The remainder of this paper describes our data collection and analysis methodology, including: related work in the areas of system dependability and failure data analysis, background information about Windows crash data and the data collection process, crash data analysis and results, a discussion of the merits of potential extensions to our work, and a conclusion.

Related Work

Jim Gray’s work [Gra86, Gra90] serves a model for most contemporary failure analysis work. Gray did not perform root cause analysis but rather Outage Cause that considers the last in the fault chain. In 1989, he found that the major source of outages was software, contributing about 55%, far outrunning its immediate

successor, system operations, which contributed 15%. This observation led him to blame software for almost every failure. In an earlier study [G05, GP05], we analyzed Windows application crashes to understand causal relationships in the user-level. Departing from Gray's outage cause analysis, in our study we perform root cause analysis under the assumption that the first crash in a sequence of crashes is responsible for all subsequent crashes within that event chain.

The past two decades have produced several studies in root-cause analysis for operating systems (OS) ranging from Guardian OS and Tandem Non-Stop UX OS to VAX/VMS and Windows NT [Gra90, Kal98, LI95, SK+00, SK+02, TI92, TI+95]. In server environments, Tandem computers, VAX clusters as well as several operating systems and file servers have been examined for software defects by several researchers. Lee and Iyer focussed on software faults in the Tandem GUARDIAN operating system [LI95], Tang and Iyer considered two VAX clusters running the VAX/VMS operating system [TI92], and Sullivan and Chillarege examined software defects in MVS, DB2, and IMS [SC91]. Murphy and Gent also focussed on system crashes in VAX systems over an extended period, almost a decade [MG95]. They concluded that system management was responsible for over 50% of failures with software trailing at 20% followed by hardware that is responsible for about 10% of failures.

While examining NFS data availability in Network Appliance's NetApp filers, Lancaster and Rowe attributed power failures and software failures as the largest contributors to downtime; operator failure contributions were negligible [LR01]. Thakur and Iyer examined failures in a network of 69 SunOS workstations [TI96]. They divided problem root causes into network, non-disk and disk-related machine problems. Kalyanakrishnam, et al. perused six months of event logs from a LAN comprising of Windows NT workstations that delivered emails [KK+99]. Using a state machine model of detailed system failure states to describe failure timelines on a single node, they concluded that most automatic system reboot problems are software-related; the average downtime is two hours. Similarly, Xu, et al. considered Windows NT event log entries related to system reboots for a network of workstations that were used for enterprise infrastructure, allowing operators to annotate event logs to indicate the reason for reboot [XK+99].

In this progression, our study of Windows' crash data gauges the evolution of PC reliability. Koopman, et al. test operating systems against the POSIX specification [KD00]. Our study is complementary to this work as we consider actual crash data that leads to OS unreliability.

Recently, in Windows XP Machines, Murphy deduced that display drivers were a dominant crash cause and memory is the most frequently failing hardware component [Mur04]. We extend this work by studying actual crash instances experienced by users

rather than injecting artificial faults as performed by fuzz testing [FM00]. Our study of crash data differs from error log analysis performed by Kalakech, et al. [KK+04]; we determine the cause of crashes in addition to time and frequency.

Several researchers have provided insights on benchmarking and failure data analysis [BC+02, BS97, OB+02, WM+02]. Wilson, et al. suggest evaluating the relationship between failures and service availability [WM+02]. Among other metrics, when evaluating dependability, system stability is a key concern. Ganapathi, et al. examine Windows XP registry problems and their effect on system stability [GW+04]. Levendel suggests using the catastrophic nature of failures to evaluate system stability [Lev89]. Brown, et al. provide a practical perspective on system dependability by incorporating users' experience in benchmarks [BC+02, BS97]. In our study of crashes, we consider these factors when evaluating various applications.

Overview of Crashes and Crashdumps

A crash is an event caused by a problem in the operating system (OS) or application (app) requiring OS or app restart. App crashes occur at user level and typically involve restarting the crashing application. An OS crash occurs at kernel-level, and is usually caused by memory corruption, bad drivers or faulty system-level routines. OS crashes are more frustrating than application crashes as they require the user to kill and restart the Windows Explorer process at a minimum, more commonly forcing a full machine reboot. While there are a handful of crashes due to memory corruption and other common systems problems, a majority of these OS crashes are caused by device drivers. These drivers are related to various components such as display monitors, network and video cards.

Upon each OS crash or bluescreen generated by the operating system, Windows XP collects failure data as a minidump. Users have three different options for the amount of information that is collected upon a crash. We use the default (and smallest) option of collecting small dumps, which are only 64K in size. These small minidumps contain a partial snapshot of the computer's state at the time of crash. They include a list of loaded drivers, the names and timestamps of binaries that were loaded in the computer's memory at the time of crash, the processor context for the stopped process, and process information and kernel context for the stopped process and thread as well as a brief stack trace. We do not collect personal data files for our study. However, portions of such data may be resident in memory at the time of crash and will consequently appear in our crash dumps. To disable personal data inadvertently being sent, crash reporting may be disabled or the user can choose not to send a particular crash report.

When an OS crash occurs, typically the entire machine must be rebooted. Any relevant information

that can be captured before the reboot is saved in a .dmp file in the %windir%\Minidump directory. These minidumps are uniquely named with the date of the crash and a serial number to eliminate conflicting names for multiple crashes on the same day.

Overview of BOINC Crash Collector

Berkeley Open Infrastructure for Network Computing (BOINC) is a platform for pooling computer resources from volunteers to collect data and run distributed computations [And03]. A popular example of an application using this platform is SETI@home, which aggregates computing power to 'search for extraterrestrial intelligence.' BOINC provides services to send and receive data from its users via the HTTP protocol using XML formatted files. It allows application writers to run and maintain a server that can communicate with numerous client machines through a specified Applications Programmer Interface (API). Each subscribed user's machine, when idle, is used to run BOINC applications. Project groups can create project web sites with registration services for users to subscribe and facilitate a project. The web site can also display statistics for contributing users.

Taking advantage of these efforts, we have created a data collection application to run on this platform. BOINC provides a good opportunity to collect and aggregate data from users outside our department while addressing privacy concerns. BOINC anonymizes user information while allowing us to correlate data from the same user. We have written tools to read minidumps from users' machines and send the data to our BOINC server. The drawback of this mechanism is that we can only collect crash dumps that are stored in known locations on the user's computer, consequently excluding application crash dumps that are stored in unknown app-specific locations. Furthermore, configuring the BOINC server is a tedious and meticulous task. We must also monitor the number of work units we allot for the BOINC projects; if there are not enough work units, the application will not run on client machines.

An attractive aspect of using BOINC is that we can add more features to our application as and when necessary. We can also provide users with personalized feedback pages, consequently rewarding the users with an incentive for sharing data. However, we must verify the integrity of each crashdump we receive from the users; users often create files in the crashdump directory to inflate their crash contribution ranking.

We use a combination of Microsoft's analysis tools and custom-written scripts to parse, filter and analyze the crash data. Received crash dumps are parsed using Microsoft's "Debugging Tools for Windows" (WinDbg), publicly available at <http://www.microsoft.com/whdc/devtools/debugging/default.mspx>. We retrieve debugging symbols from Microsoft's publicly available symbol server (<http://www.microsoft.com/whdc/devtools/debugging/symbolpkg.mspx>). Parsing crash

dumps using WinDbg reveals the module that caused the crash as well as the proximate cause of the crash via an error code of the crashing routine. The drawback of this approach is that we rely on the completeness and accuracy of Microsoft's symbols. For legal reasons, Microsoft does not make third party debugging symbols available, especially those related to antivirus and firewall software.

We have conducted experiments and noted that 10% of crashdumps parsed with publicly available debugging symbols have different analysis results as compared to results when parsed with Microsoft's internal symbols. Microsoft-written components such as ntoskrnl take the blame for several third party and antivirus/firewall-related crashes.

Once crash dumps are parsed by WinDbg, the importance of filtering data is evident. When a computer crashes, the application or entire machine is rendered unstable for some time during which a subsequent crash is likely to occur. Specifically, if a particular piece of hardware is broken, or part of memory is corrupt, repeated use is likely to reproduce the error. It is inaccurate to double-count subsequent crashes that occur within the same instability window. To avoid clustering unrelated events while capturing all related crash events, we cluster individual crash events from the same machine based on temporal proximity of the events. The data that is collected can be used to gather a variety of statistics. We can provide insight to the IT team about the dominant cause of crashes in the organization and how to increase product reliability. We can also use crash behavior to track any potential vulnerability as frequent crashes may be a result of malware on the machine. In the long run, we may be able to develop a list of safe and unsafe hardware and software configurations and installation combinations that result in crashes.

Understanding Crash Data

To study a broad population of Windows users, we studied data from public-resource computing volunteers. Numerous people enthusiastically contribute data to projects on BOINC rather than corporations as they favor a research cause. Additionally, users appreciate incentive either through statistics that compares their machine to an average BOINC user's machine, or through recognition as pioneering contributors to the project.

Currently, we have about 3500 BOINC users signed up to our project. Over the last year, we have received 2528 OS crashes from 617 of these users; several users experienced (and reported) multiple OS crashes while a majority of them reported zero or one crash. Users reporting no crashes most likely do not actively run the BOINC client on their machine.

According to results shown in Figure 1 most users experienced (submitted) only one crash; however, several users suffered multiple OS crashes. One

user appears to have experienced over 200 OS crashes over the last year! The number is staggering considering that this data is for kernel-level crashes. Perhaps the user's user-mode crash counts are as bad, if not worse, considering there is more opportunity for variability in user-mode components.

First we analyze each crash as a unique entity to determine statistics on what components cause the Windows OS to crash often. Then, to understand how crashes on the same machine relate to each other, we carefully examined machines that experienced more than 5 kernel crashes within a 24 hour time period. In several cases, we observed the same crash occurring repeatedly (i.e., same fault in same module). There were also scenarios with crashes in various components interleaved with one another. We examine user behavior, temporal patterns and device driver software reliability to understand these crashes.

A Human Perspective

The human user plays a huge role in the wear and tear of a computer. User-interaction is among the most difficult patterns to quantify. We extracted three distinct user-scenarios from examining crash sequences from our data:

- Case 1: The user retries the same action repeatedly, and consequently experiences the same crash multiple times. He believes the repetition will eventually resolve the problem (which may be true over a long period of time). In this scenario, the user's model of how things work is incomplete. He does not understand the complex dependencies within the system.
- Case 2: There is some underlying problem at a lower level that is causing various different crashes. For example, if the user has hardware problems, he is likely to have many more crashes in random components. In this case, the

user is simply flustered with all the crashes and fixing each driver involved in each crash still will not resolve his problem; he will have to fix the root cause.

- Case 3: The user knows what the problem is and simply does not see an incentive to fixing it. For example, he might be using an old version of a driver for which an update is available. There are three conceivable explanations for not updating the crashing driver: a) fear of breaking other working components, b) laziness, and c) fear of getting caught with an illegal copy of software.

A Temporal Perspective

There are factors beyond end user behavior that demonstrate inter-crash relationships. Figure 2 shows a distribution of the uptime between a machine reboot and a crash event. We observe that 25% of crashes occur within 30 minutes of rebooting a machine. 75% of crashes occur within a day of rebooting a machine. Perhaps shorter system uptime intervals indicate the trend of several consecutive related crashes.

Upon analyzing crash sequences on various machines, we observed various distinct temporal indicators of crash cause:

- <5 minute uptime: A crash that occurs within 5 minutes of rebooting a computer is most indicative of a boot-time crash. The crash is not likely to have been caused by a user action. These crashes are the most frustrating as there is very little the user can do between the time of reboot and the time of crash. The user may gain insight on such crashes by examining the boot log.
- 5 minutes-1 hour uptime: These crashes are more likely to be caused by a specific sequence of events initiated by the user (e.g., accessing a particular file from a corrupt disk segment).

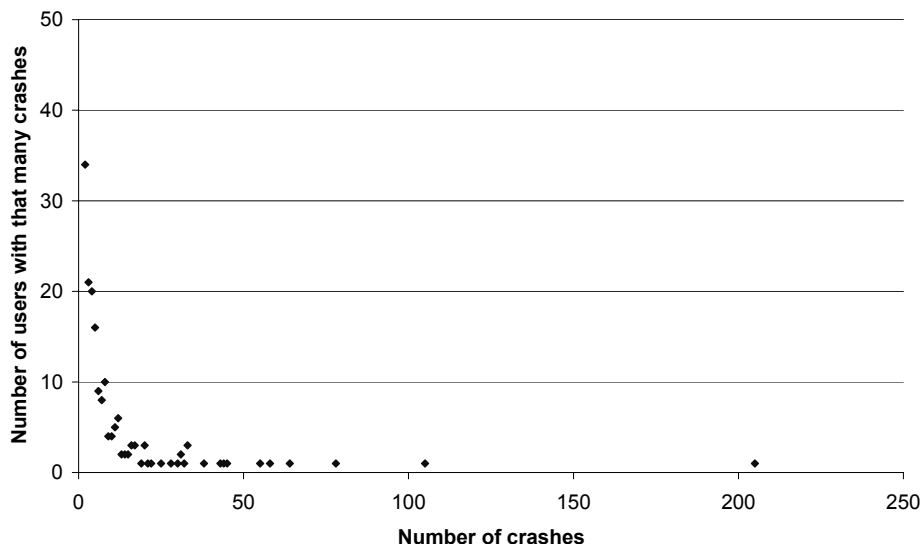


Figure 1: A histogram of the number of crashes experienced by users over the last year. One data point was omitted from the graph for clarity (443 users experienced only 1 crash each).

They could be attributed to software problems, hardware problems or memory corruption.

- **Regular interval between crashes:** Several users experienced crashes regularly at a particular time of day. Such crashes may be attributed to a periodic process resembling a cron job or an antivirus scan.
- **Context-based:** Various crashes are triggered by a logically preceding event. For example, every time a virus scanner runs, we may observe a failed disk access. In such scenarios, we cannot use exact time as an indicator.
- **Random:** Many crash sequences on users' machines did not fit in any of the above profiles. Several consecutive seemingly unrelated crashes could suggest a hardware problem and/or memory corruption.

Temporal crash patterns are useful in narrowing down a machine's potential root cause problems. However, the underlying responsibility of causing the crash lies in the longevity and reliability of the hardware and software on the machine.

A Device Driver Reliability Perspective

Device drivers are a major contributor of kernel-level crashes. A device driver is a kernel-mode module that communicates operating system requests to the device and vice versa. These drivers are inherently complex in nature and consequently difficult to write. Among many reasons for device driver complexity are that these drivers deal with asynchronous events. Since they interact heavily with the operating system, the code must follow kernel programming etiquette (which is difficult to master and follow). Furthermore, once device drivers are written, they are exceedingly

difficult to debug as the typical device driver failure is a combination of an OS event and a device problem, and thus very difficult to reproduce (see [SM+04] for a detailed description of device driver problems).

Figure 3 is largely based on the OS Crash Type field in analyzed crash reports. This field reveals graphics driver faults, common system faults (such as memory/pool corruption and hardware faults) and Application faults. However, there were many instances where the OS Crash Type was not provided (or defaulted to "Driver Fault") for legal reasons. In the absence of details revealed by the analysis tools, we crawled the web to derive the type of each driver that caused a crash. Where we were unable to determine the driver type (for example, when the documentation was not in English), we defaulted to "unknown."

Figure 4 shows that a handful of organizations contribute a significant number of crash-causing drivers to our data. Drivers written by seven organizations (Microsoft, Intel, ATI Technologies, Nvidia, Symantec, Zone Labs and McAfee) contributed 75% of all crashes in our data set. This trend suggests that crashes caused by poorly-written and/or commonly used drivers can be reduced significantly by approaching these top seven companies. On the other hand, the graph has a heavy tail, indicating that it would be extremely difficult to eliminate the remaining 25% of crashes as they are caused by drivers written by several different organizations.

Subsequently, we study the image (i.e., .exe, .SYS, or .dll file) that caused these crashes and identify the organization that contributed the crash-causing code see Figure 5.

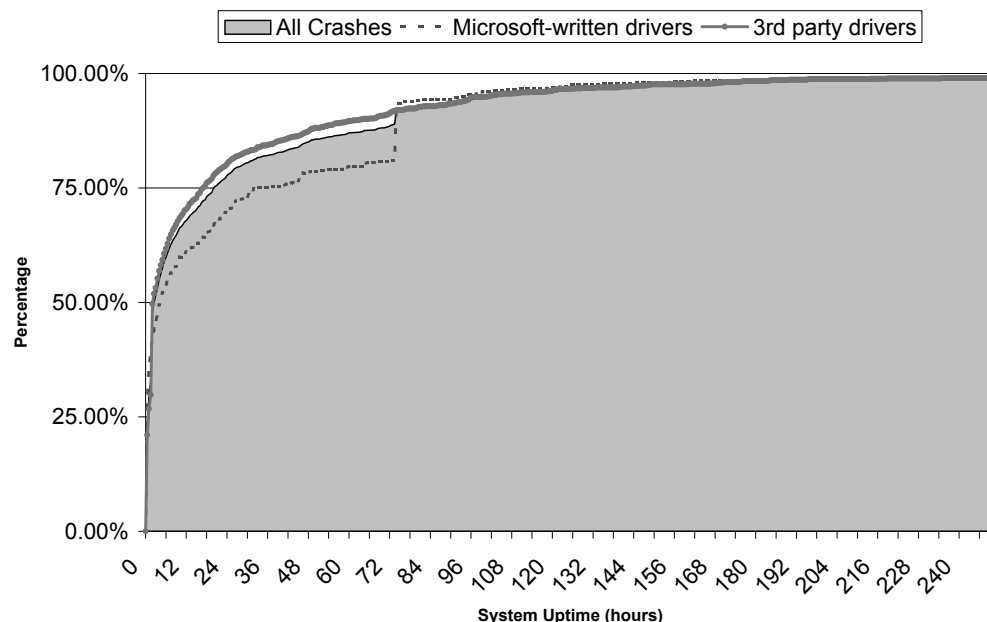


Figure 2: A cumulative frequency graph of system uptime between reboot and crash events. The dotted line extrapolates what the CFG would look like if Microsoft wrote all the drivers while the dashed line suggests what the CFG would look like if Microsoft wrote none of the drivers that crashed.

The top contender in Figure 5 is ialmdev5.dll, the Intel graphics driver. Recently, graphics drivers have become notorious for causing crashes and ialmdev5.dll is perhaps one of the more commonly used drivers in this category due to the popularity of Intel processors.

The second highest contender in Figure 5 is ntoskrnl.exe, which constitutes the bare-bones Windows NT operating system kernel code. It is not surprising that this executable is responsible for a number of driver crashes because it interacts with every other operating system component and is thus the single most critical component that can never be perfect

enough. Furthermore, other systems code might generate bad input parameters to the ntoskrnl functions that cause exceptions; ntoskrnl bears the blame for the resulting crash as it generated the exception. Also, as mentioned earlier, many antivirus/firewall-related crashes may have been mis-categorized, blaming ntoskrnl due to third party privacy concerns (hence the significantly high percentage of crashes attributed to Microsoft in Figure 3).

Other crash causing images range from I/O drivers to multimedia drivers. It is difficult to debug or even analyze these crashes further as we do not have the code and/or symbols for these drivers.

OS CRASH TYPE	NUMBER OF CRASHES	OS CRASH TYPE	NUMBER OF CRASHES
OS Core	726	Networking	338
Microsoft	488	Unknown	194
Unknown	238	Microsoft	51
Graphics Drivers	495	Conexant	17
Intel	287	Other	76
ATI Technologies	97	Common System Fault (Hardware and Software Memory Corruption)	136
Nvidia	67	Audio	130
Other	44	Avance Logic	44
Application Drivers	482	C-Media	33
Intel	89	Microsoft	16
Microsoft	64	Other	37
Symantec	58	Storage	106
McAfee	55	Microsoft	82
Zone Labs	55	Other	24
Unknown	13	Other	95
Other	148	Unknown	20

Figure 3: Number of OS crashes of each type based on 2528 crashes received from BOINC users. (We would need many more samples before it would be safe generalizing these results to a larger user community.) This table also shows the top few crash-causing driver writers in each category.

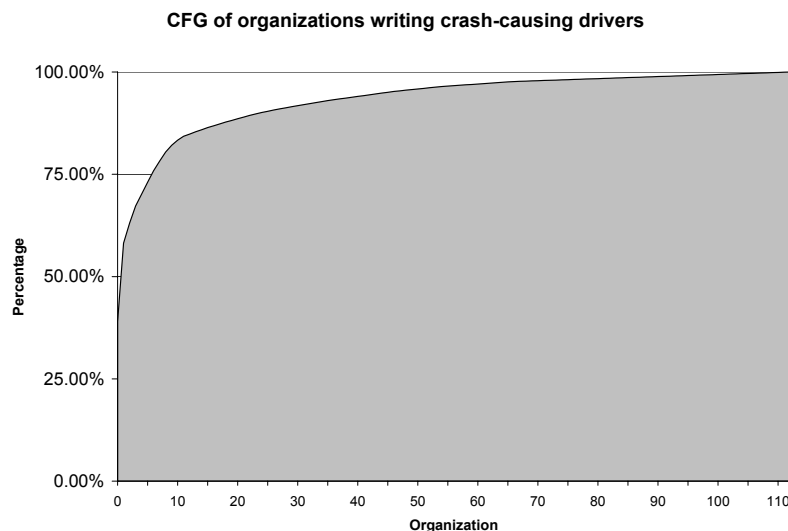


Figure 4: Cumulative Frequency Graph of organizations responsible for crash-causing drivers in our data. This graph does not account for driver popularity. 113 companies are represented in this graph.

With the increasing need for numerous devices accompanying the PC, it does not scale for the operating system developers to account for and write device driver code for each device; consequently, device drivers are written by device manufacturers, who are potentially inexperienced in kernel programming. Perhaps such lack of expertise is the most impacting cause for driver-related OS crashes.

We also observed numerous OS crashes caused by memory corruption. Memory corruption-related crashes can often be attributed to hardware problems introduced by the type of memory used (e.g., non-ECC memory). In the event that the memory corruption was due to software, the problem cannot be tracked down to a single image.

To further understand driver crashes, we studied the type of fault that resulted in the crash. Figure 6 lists the number of crashes that were caused by the various fault types. These fault types are reported by Microsoft's analysis tools when analyzing each OS crash dump.

While many of these fault types are straightforward to understand from the name, many others are abbreviations of the event they describe. Below, we enumerate each fault type and its significance (based on the descriptions provided in the parsed crash dumps):

- **IRQL NOT LESS OR EQUAL** – An attempt was made to access a pageable (or completely invalid) address at an interrupt request level (IRQL) that is too high. The driver is most likely using an improper address.¹
- **THREAD STUCK IN DEVICE DRIVER** – The device driver is spinning in an infinite loop, most likely waiting for hardware to become idle. This usually indicates problem with the hardware itself or with the device driver programming the hardware incorrectly.

¹The interrupt request level is the hardware priority level at which a given kernel-mode routine runs, masking off interrupts with an equivalent or lower IRQL on the processor. A routine can be preempted by an interrupt with a higher IRQL.

Image Name/ Crash Cause	Image Description	Num Crashes	% Crashes	% Running Total
lalmdev5.DLL	Intel graphics driver	275	11%	11%
ntoskml.exe	NT kernel and system	187	8%	19%
CAPI20.SYS	ISDN modem driver	182	7%	26%
Win32k.sys	multi user win32 driver	114	5%	31%
IdeChnDr.sys	Intel Application Accelerator driver	89	4%	35%
ntkrnlmp.exe	Multi-processor version of NT kernel and system	87	4%	39%
vsdatant.sys	TrueVector Device Driver	51	2%	41%
GDFSHK.SYS	McAfee Privacy Service File Guardian	48	2%	43%
V7.SYS	IBM V7 Driver for Windows NT/2000	45	2%	45%
ALCXWDM.SYS	Windows WDM driver for Realtek AC'97	44	2%	47%

Figure 5: Top 10 OS Crash-causing Images based on 2528 crashes received from BOINC users. (We would need many more samples before it would be safe generalizing these results to a larger user community.) A description of the crash-causing image is provided in addition to the percentage of crashes caused by each image.

Driver Fault Type	Num Crashes
IRQL NOT LESS OR EQUAL	657
THREAD STUCK IN DEVICE DRIVER	327
PAGE FAULT IN NONPAGED AREA	323
KERNEL MODE EXCEPTION NOT HANDLED	305
UNEXPECTED KERNEL MODE TRAP	78
BAD POOL CALLER	74
SYSTEM THREAD EXCEPTION NOT HANDLED	73
PFN LIST CORRUPT	53
DRIVER CORRUPTED EXPOOL	38
MACHINE CHECK EXCEPTION	37

Figure 6: Top 10 crash generating driver fault types.

- **PAGE FAULT IN NONPAGED AREA** – Invalid system memory was referenced, for example, due to a bad pointer.
- **KERNEL MODE EXCEPTION NOT HANDLED** – The exception address pinpoints the driver/function that caused the problem. However, the particular exception thrown by the driver/function was not handled.
- **UNEXPECTED KERNEL MODE TRAP** – A trap occurred in kernel mode, either because the kernel is not allowed to have/catch (bound trap) the trap or because a double fault occurred.
- **BAD POOL CALLER** – The current thread is making a bad pool request. Typically this is at a bad IRQL level or double freeing the same allocation, etc.
- **SYSTEM THREAD EXCEPTION NOT HANDLED** – This fault type is similar to an unhandled kernel mode exception.
- **PFN LIST CORRUPT** – Typically caused by drivers passing bad memory descriptor lists.
- **DRIVER CORRUPTED EXPOOL** – An attempt was made to access a pageable (or completely invalid) address at an interrupt request level (IRQL) that is too high. This fault is caused by drivers that have corrupted the system pool.
- **MACHINE CHECK EXCEPTION** – A fatal machine-check exception occurred (due to hardware).

Studying these fault types reveals various programming errors that impact system behavior and what OS problems to tackle with caution. However, this information is more useful to the software developer than the end user. From a user's perspective, the most useful piece of information is "what can I fix on my machine?"

There are three distinct trends we observed on machines with multiple crashes:

- **The same driver causes most crashes:** This scenario is very simple to resolve. Most likely, the crash-causing driver is an old version, which has newer, more stable version available. There were other cases where a newly downloaded driver caused various crashes as a result of its incompatibility with other components installed on the machine. In both situations, updating or rolling back the driver's version will reduce crashes on the machine.
- **Related drivers cause most crashes:** Two drivers are considered related if they communicate with the same device or pertain to the same component. In this scenario if different, yet related, drivers cause the machine's crashes, then perhaps the common underlying component or device is at fault and needs attention.
- **Unrelated drivers cause the crashes:** This scenario is the most difficult to comprehend. First, we understand what the drivers have in common – whether they perform similar actions or

function calls, have similar resource requirements (e.g., requiring network connectivity), or access the same objects.

In the above scenarios, it is useful to understand inter-driver dependencies. We would also benefit from understanding the stability of specific driver versions and how diverse their install base is.

Discussion

Windows users have started viewing crashes as a fact of life rather than a problem. We have the single most valuable resource to design a system that helps users cope with crashes better – crash data. Microsoft's Online Crash Analysis provides users with feedback on each of their submitted crashes. However, many users suffer from multiple crashes and individual per-crash analysis is not enough to identify the optimal solution to the root problem. There is a strong need to use historical data for each machine and use heuristics to determine the best fix for that machine.

The human, temporal and device-driver reliability perspectives shed light on potential root causes for crashing behavior. There are numerous other factors we can include to refine root cause analysis. It would be very beneficial to scrape portions of the machine's event log when analyzing crashes. We can look for significant events preceding each crash (e.g., Driver installed/removed, process started up, etc.), pinpointing likely sources of the machine's behavior.

It is also useful to collect various machine health metrics such as frequency of prophylactic reboot and frequency of virus scans. Such metrics will help us evaluate the relative healthiness of a machine (compared to the entire user population) and customize analysis responses on a per-machine basis. Ideally we would want our data analysis system to have a built-in feedback loop (as seen in Figure 7) so we can continuously adapt and improve our analysis engine. This framework is useful for performing accurate post-mortem analysis.

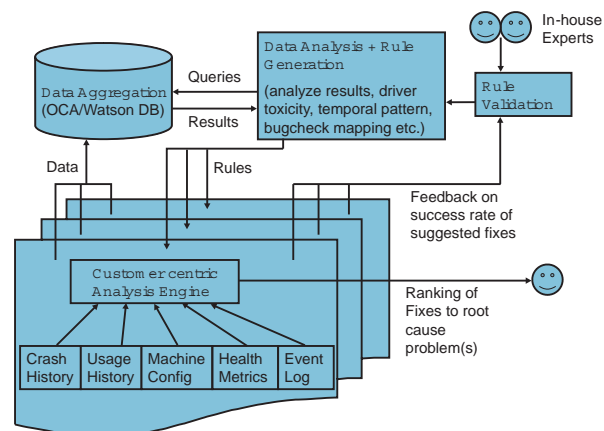


Figure 7: Customer-centric kernel crash analysis framework.

It is equally important to understand the manifestation of such problems on each machine. It is important to characterize inter-component interactions and model failure propagation patterns. Such analysis will help improve inter-component isolation, reducing their crash-likeness. While post-mortem analysis and debugging helps cure problems, it is also critical to prevent problems at their source. As an industry, we must work towards determining the characteristics of software that dictate software dependability.

Conclusion

Our crash-data related study, despite the small quantity of Windows XP data analyzed, has contributed several observations. The most notable reality is that the Windows operating system is not responsible for a majority of PC crashes in our data set. Poorly-written device drivers contribute most of the crashes in our data. It is evident that targeting a few companies to improve their driver quality will effectively eliminate 75% of our crashes. However, the remaining 25% of crashes are extremely difficult to eliminate due to the large number of organizations contributing the driver code.

Users can alleviate computer frustration by better usage discipline and avoiding unsafe applications and drivers. With additional data collection and mining, we hope to make stronger claims about applications and also extract safe product design and usage methodology that apply universally to all operating systems. Eventually, this research can gauge product as well as usage evolution.

Studying failure data is as important to the computing industry as it is to consumers. Product dependability evaluations help evolve the industry by reducing quality differential between various products. Once product reliability data is publicized, users will use such information to guide their purchasing decisions and usage patterns. Product developers will react defensively and resulting competition will improve quality control.

In the future, we hope to refine analysis engine and automate many of the background queries for each driver. We would like to improve our understanding of the dependencies between analysis categories such as the temporal and device driver perspectives. We also plan to investigate the relationship of various objects involved at the time of crash. Lastly, we would like to obtain more environmental metrics and draft more rules for analysis, and extend this work to other Operating Systems.

Author Biographies

Archana Ganapathi is a graduate student at the University of California at Berkeley. She completed her Masters degree in 2005 and is currently pursuing her Ph.D. in Computer Science. Her primary research interests include operating system dependability and system management.

Viji Ganapathi is an undergraduate at the University of California at Berkeley. She will complete her Computer Science Bachelors degree in December, 2006.

David A. Patterson has been Professor of Computer Science at the University of California, Berkeley since 1977, after receiving his A.B., M.S., and Ph.D. from UCLA. He is one of the pioneers of both RISC and RAID, both of which are widely used. He co-authored five books, including two on computer architecture with John L. Hennessy. They have been popular in graduate and undergraduate courses since 1990. Past chair of the Computer Science Department at U.C. Berkeley and the Computing Research Association, he was elected President of the Association for Computing Machinery (ACM) for 2004 to 2006 and served on the Information Technology Advisory Committee for the U.S. President (PITAC) from 2003 to 2005.

His work was recognized by education and research awards from ACM and IEEE and by election to the National Academy of Engineering. In 2005 he shared Japan's Computer & Communication award with Hennessy and was named to the Silicon Valley Engineering Hall of Fame. In 2006 he was elected to the American Academy of Arts and Sciences and the National Academy of Sciences and he received the Distinguished Service Award from the Computing Research Association.

Bibliography

- [And03] Anderson, D., "Public Computing: Reconnecting People to Science," *The Conference on Shared Knowledge and the Web*, Residencia de Estudiantes, Madrid, Spain, Nov., 2003.
- [BS+02] Broadwell, P., N. Sastry and J. Traupman, "FIG: A Prototype Tool for Online Verification of Recovery Mechanisms," *Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN)*, New York, NY, June, 2002.
- [BC+02] Brown, A., L. Chung, and D. Patterson, "Including the Human Factor in Dependability Benchmarks," *Proc. 2002 DSN Workshop on Dependability Benchmarking*, Washington, D.C., June, 2002.
- [BS97] Brown, A. and M. Seltzer, "Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture," *Proc. 1997 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, Seattle, WA, June, 1997.
- [FM00] Forrester, J. and B. Miller, "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing," *Proc. 4th USENIX Windows System Symposium*, Seattle, WA, Aug., 2000.
- [G05] Ganapathi, A. "Why Does Windows Crash?" *UC Berkeley Technical Report UCB/CSD-05-1393*, May, 2005.

- [GW+04] Ganapathi, A., Y. Wang, N. Lao and J. Wen, "Why PCs are Fragile and What We Can Do About It: A Study of Windows Registry Problems," *Proc. International Conference on Dependable Systems and Networks (DSN-2004)*, Florence, Italy, June, 2004.
- [GP05] Ganapathi, A. and D. Patterson, "Crash Data Collection: A Windows Case Study," To Appear in *Proc. International Conference on Dependable Systems and Networks (DSN-2005)*, Yokohama, Japan, June, 2005.
- [Gra86] Gray, J., "Why Do Computers Stop and What Can Be Done About It?" *Symp. on Reliability in Distributed Software and Database Systems*, pp. 3-12, 1986.
- [Gra90] Gray, J., "A census of Tandem system availability between 1985 and 1990," *Tandem Computers Technical Report 90.1*, 1990.
- [GS04] Gray, J. and A. Szalay, "Where the rubber meets the sky:bridging the gap between databases and science," *Microsoft Research TR-2004-110*, 2004.
- [KK+04] Kalakech, A., K. Kanoun, Y. Crouzet and J. Arlat, "Benchmarking the dependability of Windows NT4, 2000 and XP," *Proc. International Conference on Dependable Systems and Networks (DSN-2004)*, Florence, Italy, June, 2004.
- [Kal98] Kalyanakrishnam, M., "Analysis of Failures in Windows NT Systems," Masters Thesis, *Technical report CRHC 98-08*, University of Illinois at Urbana-Champaign, 1998.
- [KK+99] Kalyanakrishnam, M., Z. Kalbarczyk, and R. Iyer, "Failure data analysis of a LAN of Windows NT based computers," *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, 1999.
- [KD+05] King, S., G. Dunlap and P. Chen, "Debugging operating systems with time-traveling virtual machines," *Proceedings of the 2005 Annual USENIX Technical Conference*, April, 2005.
- [KD00] Koopman, P., and J. DeVale, "The Exception Handling Effectiveness of POSIX Operating Systems," *IEEE Transactions on Software Engineering*, Vol. 26, Num. 9, pp. 837-848, Sept., 2000.
- [LR01] Lancaster, L. and A. Rowe, "Measuring real-world data availability," *Proceedings of LISA 2001*, 2001.
- [LI95] Lee, I. and R. Iyer, "Software Dependability in the Tandem GUARDIAN Operating System," *IEEE Transactions on Software Engineering*, Vol. 21, Num. 5, pp. 455-467, May, 1995.
- [Lev89] Levendel, Y., "Defects and Reliability Analysis of Large Software Systems: Field Experience," *Digest 19th Fault-Tolerant Computing Symposium*, pp. 238-243, June, 1989.
- [MR+04] Maniatis, P., M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, and M. Baker, "The lockss peer-to-peer digital preservation system," *ACM Transactions on Computer Systems (TOCS)*, 2004.
- [Mur04] Murphy, B., "Automating Software Failure Reporting," *ACM Queue*, Vol. 2, Num. 8, Nov., 2004.
- [MG95] Murphy, B. and T. Gent, "Measuring system and software reliability using an automated data collection process," *Quality and Reliability Engineering International*, Vol. 11, 1995.
- [OB+02] Oppenheimer, D., A. Brown, J. Traupman, P. Broadwell, and D. Patterson, "Practical issues in dependability benchmarking," *Workshop on Evaluating and Architecting System dependability (EASY '02)*, San Jose, CA, Oct., 2002.
- [SS72] Schroeder, M. and J. Saltzer, "A Hardware Architecture for Implementing Protection Rings," *Communications of the ACM*, Vol. 15, Num. 3, pp. 157-170, March, 1972.
- [SK+00] Shelton, C., P. Koopman, K. DeVale, "Robustness Testing of the Microsoft Win32 API," *Proc. International Conference on Dependable Systems and Networks (DSN-2000)*, New York, June, 2000.
- [SK+02] Simache, C., M. Kaaniche, A. Saidane, "Event log based dependability analysis of Windows NT and 2K systems," *Proc. 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)*, pp. 311-315, Tsukuba, Japan, Dec., 2002.
- [SC91] Sullivan, M. and R. Chillarege, "Software defects and their impact on system availability – a study of field failures in operating systems," *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, 1991.
- [SM+04] Swift, M., Muthukaruppan, B. Bershad, and H. Levy, "Recovering Device Drivers," *Proceedings of the 6th ACM/USENIX Symposium on Operating Systems Design and Implementation*, San Francisco, CA, Dec., 2004.
- [TI92] Tang, D. and R. Iyer, "Analysis of the VAX/VMS Error Logs in Multicomputer Environments – A Case Study of Software Dependability," *International Symposium on Software Reliability Engineering*, Research Triangle Park, North Carolina, Oct., 1992.
- [TI96] Thakur, A. and R. Iyer, "Analyze-NOW-an environment for collection and analysis of failures in a network of workstations," *IEEE Transactions on Reliability*, Vol. R46, Num. 4, 1996.
- [TI+95] Thakur, A., R. Iyer, L. Young and I. Lee, "Analysis of Failures in the Tandem NonStop-UX Operating System," *International Symposium on Software Reliability Engineering*, Oct., 1995.
- [WL+93] Wahbe, R., S. Lucco, T. Anderson, and S. Graham, "Efficient Software-Based Fault Isolation," *Proc. Fourteenth ACM Symposium on*

- Operating Systems Principles (SOSP)*, pp. 203-216, December, 1993.
- [WC+01] Welsh, M., D. Culler and E. Brewer, "SEDA, an Architecture for well-conditioned scalable Internet Services," *18th Symposium on Operating System Principles*, Chateau Lake Louise, Canada, October, 2001.
- [WM+02] Wilson, D., B. Murphy and L. Spainhower, "Progress on Defining Standardized Classes for Comparing the Dependability of Computer Systems," *Proc. DSN 2002 Workshop on Dependability Benchmarking*, Washington, D.C., June, 2002.
- [XK+99] Xu, J., Z. Kalbarczyk and R. Iyer, "Networked Windows NT system field failure data analysis," *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing*, 1999.

SUEZ: A Distributed Safe Execution Environment for System Administration Trials

Doo San Sim and V. N. Venkatakrishnan – University of Illinois, Chicago

ABSTRACT

In this paper, we address the problem of safely and conveniently performing “trial” experiments in system administration tasks. System administrators often perform such trial executions that involve installing new software or experimenting with features of existing software. Often such trials require testing of software that run on multiple hosts. For instance, experimenting with a typical client-server application requires understanding the effect of the actions of the client program on the server. We propose a distributed safe execution environment (DSEE) where such tasks can be performed safely and conveniently. A DSEE performs one-way isolation of the tasks run inside it: the effects of the client and the server are prevented from escaping outside the DSEE, and therefore are prevented from interfering with the processes running outside the DSEE. At the end of the trial execution, a DSEE allows clear inspection of the effects of running the task on all the hosts that are involved in the task execution. Also, a DSEE allows the changes to the “committed,” in which case the actions become visible outside the DSEE. Otherwise, they can be “aborted” without affecting the system in any way. A DSEE is an ideal platform through which a system administrator can perform such trials without the fear of damaging the system in any manner. In this paper, we present the design and implementation of a tool called SUEZ that allows a system administrator to create and use distributed safe execution environments. We have experimented with several client-server applications using our tool. By performing these trials in a DSEE, we have found configuration vulnerabilities in our trials that involve some commonly used client-server applications.

Introduction

System administrators and desktop users encounter various situations in their day-to-day activity that require them to download, install and run applications on their machines. One of the most common tasks is that of a “trial” run of a piece of software that the administrator. Such a trial is typically done if a system administrator has no prior experience in using that piece of software, but there are several other reasons for such a trial execution.

- *Understanding actions of a program.* Often, system administrators would like to study the impact of executing a particular command on their system. More importantly, they often would like to exercise a particular option in a program, and see the observable effects of exercising that option. For instance, when exercising an option in a particular program, the administrator would like to know the direct and indirect effects of using that option. The abundance of binary programs and programs equipped with graphical user interfaces (as opposed to script based installations) often compound this difficulty, as a lot of critical system changes happen “behind the scenes.”
- *Testing compatibility with existing configurations* Often a system administrator wonders whether installation of an application will work (co-operatively) with existing packages and

configurations. Another issue she is concerned about is about the security of user data that is handled by the application, and whether the data handled by the application is adequately protected through file permissions.

- *Experimenting with new software.* Often users download freeware/shareware from various Internet sources. These software may be untrusted or faulty and hence it is important to understand the effects of these software. Hence, system administrators may wish to perform several walk-throughs of these tools to ensure that they do not create any new problems related to security and/or interoperability.
- *Patch testing.* Application of software patches and updates too early may leave the system with potential interoperability issues created due to the updates. Another issue is with possible bugs in the patches/updates. (This is usually the reason updates are delayed much).

Often the user¹ performs the above tasks while facing the need for an environment that allows convenient study of the impact of such tasks. By impact on the system, we refer to issues related to general operation, interoperability with existing applications and security.

While the goal is to understand the actions of a program in a networked system, we note that users are

¹In this paper, we use the terms *user* and *system administrator* interchangeably, unless otherwise mentioned.

not interested in every action of a program, but only those actions whose effects they perceive as relevant to system interoperability and security. This requires that we abstract away from internal actions of a program, (such as function calls and assignments to local variables), and focus on *observable* actions of a program. Some examples of such actions are a) addition of new users b) modification to user files c) changes to local configuration files d) changes to boot time scripts.

To understand the impact of such changes on a host, *Safe Execution Environments* (SEE) were proposed in [15, 23] and for the Windows platform in [25]. A SEE uses *one-way isolation* to effect containment of the tasks run inside the SEE. Processes running outside the SEE do not see the changes made by the tasks run inside the SEE. At the end of execution, one can examine the changes made to the SEE environment, and decide whether to keep them or discard them and return to the original state.

A SEE is a highly effective environment to perform system administration trials that involve a single host. However, for tasks that are distributed over a set of networked hosts it is not directly suitable. A typical example is a client-server application where any action triggered by a client may change the system state on the server host. In this case, to understand the changes on the server that were triggered by the actions of the client, we need a *distributed* environment. This paper presents the design and implementation of a SUEZ, a tool that allows for creating distributed safe execution environments (DSEE) to assist in system administration trials.

Let us consider a simple system administration example that involves remote administration of printer software. The Common UNIX Printing System (CUPS) [1] allows remote administration of printers using a specialized port on the printer server (TCP port 631). Now, on loading the printer web interface page from the print server, the user is presented with several options related to adding and managing printers and jobs. Each of these options triggers a specific change in the printer server. For instance, adding a printer requires changes in the server on the printer driver file `/etc/cups/ppd`, and changes to the `/etc/printcap` that lists the printers. All these changes take effect when the user executes the command to add a printer. In order to know the specific changes made by a command, the system administrator is left with two options. The first one is to read manuals and other forms of documentation. The second one is the use of low level system tools. While some may argue that these are viable options in the case of a well-known application such as CUPS, they are unsuitable in the case of new/experimental software, software updates and patches.

Thus, understanding the key impacts of installing a software package/patch requires the following abilities:

1. To make the observable effects of an action on a host transparent to the user: In the above example, the action is the choice selection (through the menu displayed by the browser) to add a printer.
2. To make transparent the observable effects of these actions on other hosts in the network: This corresponds to changes to the files `/etc/cups/ppd` and `/etc/printcap` in the printer server.
3. To see the “difference” between the state of the system in all affected hosts before the action and after it: For the above example, this requires us to identify the above-mentioned files before and after the add-printer action, and any changes to system objects in the filesystem client (in this case there are none).
4. To correlate the above three to arrive at a complete understanding of the actions of the software under scrutiny: This requires us to log the temporal sequence of actions performed on both the client and server in a unified view.
5. In the event of the system administrator is not satisfied with the results, restore the state of the system to that before that of the start of installation (i.e., undo the effect of observable actions). The un-doing capability is needed to perform any experimentation on real systems.

Related Work

In this section, we discuss related work that are available as options to the system administrator. We first state the requirements of any system that would satisfy our objectives 1) to 5) given above.

1. *Allow the task to execute to completion.* In order to study the effects of a trial execution, we must allow the application to execute to completion. This will ensure that the results of a trial execution match the results of a real execution when the application is actually installed and deployed.
2. *Track the effect of the task on multiple hosts.* During execution, a task may further trigger changes to system objects in other hosts, as given in the CUPS example above. This suggests that any approach that addresses this problem must have support for *distributed monitoring*, thereby tracking and correlations the actions of a program on other hosts on the network.
3. *Support customizable unified logging.* The temporal sequence of operations that result in changes to the objects in various hosts need to be logged in a central location, where they can be analyzed. In addition, to focus on events of interest to the system administrator, the logging system must be simple enough to support customizable filters to reduce the size and complexity of evaluating them.

4. *Ability to undo the effects of actions of a program.* This is required to ensure that the system can be restored to its original state before the program was executed.

Below, we discuss the related work by grouping them into various categories. At the end of this section, we discuss the suitability of each approach category in matching the the above requirements.

Logging based approaches A typical approach way to understand the effects of executing a particular software is through the use of logging [2]. The system administrator can enable the logging options present in the software, and then inspect the logs after the operation to have an understanding of the actions of the software. The problem with this approach is that it is completely dependent on the developer of the software system/patch to log its actions. Thus this is not very dependable option as many software systems are written without logging features. Of course, with experimental software this approach clearly will not work. Also, an approach purely based on logging will make the job of reverting the system back to original state quite tedious, error-prone and in some cases, impossible.

Use of program tracing tools A second approach is to use tools such as *ltrace* [10] and *strace* [4] to study the actions of a piece of software. While this approach may reveal the effects of running or upgrading an application, one sees the effects of the software *after* it has finished execution, when the applications actions have already affected the system. It may be too late, as recovery actions may involve clean-up actions such as restoring files from backups, or removing user-ids created by the application. Approaches such as sandboxing [14, 13, 18, 22, 7, 19] do not work too, as they simply restrict the execution of the software, rather than allowing it to run completely in order to study its actions. Use of package managers such as RPM and dpkg may simplify the problem of uninstallation; but they do not offer any help in understanding the effects of software that are already installed. Furthermore, package managers are inapplicable if the software is distributed in binary or source forms.

VM based approaches A third approach is to use special machines [16, 12] or even virtual machines [5, 11, 24] for studying the effects of a particular piece of software. In order to correctly track the effects of the system, machines and special hardware have the problem of accurate environment reproduction, where the system configuration on the virtual machine environment needs to accurately reflect the one on the production environment. Such accurate environment reproduction is crucial to ensure that the system behavior on the VM is same as that on the production system. Another possibility is make use of snapshot features in modern virtual machines such as VMware. However, these snapshots tend to give the difference of the actions of the entire set of processes running on the system and not the programs the user wishes to focus on.

Recovery-oriented approaches Although recovery from failures is not the primary goal of our approach, we do provide facilities for recovery in case of a task failure. The Recovery-Oriented Computing (ROC) project [20] is developing techniques for fast recovery from failures, focusing on failures due to operator errors. [8] presents an approach that assists recovery from operator errors in administering a network server, with the specific example of an email server. The recovery capabilities provided by their approach are more general than those provided by ours. The price to be paid for achieving more general recovery capabilities is that their approach is application specific. In contrast, through a DSEE we provide a general task-independent framework for troubleshooting and recovery.

Discussion Note that sandboxing based approaches do not fully support the objective of allowing a task to run to completion (point a) above), as they block actions of a program based on the policy. So using sandboxing, we have no way of learning the complete effects of a piece of software. Logging based systems allow the applications to run with complete freedom, but do not support undoing of actions (point d) above). File versioning systems [17, 21] and virtual machine based snapshot approaches may satisfy undoing at a more general level, but not based on a program or specific actions of a program and therefore do not satisfy point d) above. Furthermore, they do not directly support point b) and c) above. On the other hand, executing a task in a DSEE will address all the objectives a) to d) above.

Paper Organization This paper is organized as follows. In the next section, we discuss the concept of one-way isolation that serves as the basis for our approach for building DSEEs. We then discuss the design details of our framework for building DSEEs followed by the routing enhancements to automatically provide the redirection facility for network operations. We explicate a message handling subsystem that we implemented for communication between various DSEEs. We present a system evaluation by performing various trials using our system and discuss the performance costs followed by a conclusion.

One-way Isolation

Our approach builds on the one-way isolation approach presented in [15, 23]. We briefly review the one-way isolation approach that we employ to create distributed safe execution environments (DSEE).

Isolation of a set of tasks refers to the property that disallows the effects of such tasks from being made available until its completion. In database systems, isolation is one of the ACID properties. The main objective in using isolation in our approach is to effect containment of the trial execution task performed inside the isolated environment. Any operation that is only “reads” the system (i.e., one that reads the

system state but does not write/modify it) may be performed by SEE processes. It also means that “write” operations should not be permitted to change the state of the system. There are two options to implement the environment such that isolation is achieved: one is to *restrict* the operation, i.e., disallow its execution. The second option is to *redirect* the operation to a different resource that is invisible outside the safe execution environment. To maintain the correctness of the resource access operations, it is important to maintain the redirection for subsequent operations (such as writes) from the program. Below, we discuss both restriction and redirection for performing system administration trials.

Through *restriction*, an operation initiated by a process is prevented from completion. When this happens, an exception may be returned to the process. To implement restriction, we need to know the set of operations that may affect the state of the system. However, in the context of performing trial executions, an approach purely based on restriction is not likely to be very successful as it will prevent applications from running successfully to completion. For instance, a program may intend to perform a network operation by opening and socket and listening to messages on that socket. If this operation is restricted, this program will not be able to successfully receive messages. Most non-trivial client-server applications will fail for similar reasons. Hence, in our approach we resort to restriction only if the other redirection option is not likely to provide successful results.

The other choice for implementing isolation is through redirection. In *redirection*, any operation that accesses a resource is redirected to another resource that is unavailable to the rest of the system. For instance, when a file modification operation is performed by a SEE process, a copy of the original file may be created in a “private” area of the filesystem, and the modification operation is performed on this copy. Redirection does not suffer from the same problem as restriction and the SEE process is likely to successfully run to completion under redirection.

Two forms of redirection are possible: *static* or *dynamic*. Static redirection requires the source and target objects to be specified in advance of the operation, in fact before the SEE process is executed. For instance, one may statically specify that operations to bind a socket to a port p should be redirected to an alternate port p' . Similarly, one may specify that operations to connect to a port p on host h should be redirected to host h' (which may be the same as h) and port p' . However, such static redirection becomes hard to implement when the number of possible targets is too large to be specified in advance or if a SEE process performs a large number of such operations that are distinct. For instance, it may be hard to predict the number and location of files on a server that may be accessed or modified by a client operation. Moreover,

such modification operations have indirect side effects that involve dependencies between such object, e.g., the file operations on the server involve changes to the directories these files reside in. A redirection operation that ignores the effect on these directories simply will not work. In such case, *dynamic redirection* where the target for redirection is determined dynamically during execution.

In this paper, by using such redirection, we show how to build *distributed SEEs* (DSEE), where processes executing within SEEs on multiple hosts can communicate with each other. Such distributed SEEs are particularly useful for safe execution of a network server application, whose testing would typically require accesses by nonlocal client applications. (Note, however, that this approach for distributed SEEs works only when all cross-SEE communications take place directly between the SEE processes, and not through other means, e.g., indirect communication through a shared NFS directory.)

In our current implementation, system call interposition is used to implement restriction and static redirection. We restrict all modification operations other than those that involve the file system and the network. In the case of file operations, all accesses to normal files are permitted, but accesses to raw devices and special purpose operations such as mounting file systems are disallowed.

In terms of network operations, we permit any network access that can be dynamically redirected. This entails any local network operation such as a service request from a host in the network. Dynamic redirection is currently supported in our implementation for a number of commonly used network services.

After the trial execution is over, the system administrator can examine the results of the trial execution. If the results are satisfactory, she can commit the results back to the file systems on the respective hosts that run the DSEE. Commit criteria for such executions have been developed in [23]. In this paper, we do not discuss criteria for committing. Instead, our focus is solely on construction of DSEEs and performing system administration experiments with them.

Our Approach

Figure 1 shows the a network-level overview of SUEZ. There are two main components in SUEZ that are responsible for creating a DSEE. They are a) a host level monitor that runs on each SUEZ host and b) a network redirector that runs on the main router. Each host under SUEZ has a *host monitor* component. This host monitor is responsible for isolating any local operation or remote operation. Such host-level isolation component resides on all the other hosts that are similar in the network, and the isolation environments in all these hosts collectively form a DSEE isolation context. The host monitor also runs a messaging service that it uses to communicate with other DSEEs.

The router has a component of SUEZ that performs transparent network level *host and service redirection*. The use of transparent host and service redirection allows the user of the system to run experiments without having to know the network and service requirements of the task to be performed in advance. Each host monitor logs its actions, and these logs are integrated in a log server. The log server presents the temporal sequence of operations performed during the trial execution.

Host Monitor

Figure 2 presents a detailed view of the host monitor. Each host-level monitor is built on top of the isolation module present in [15]. These monitors are used for tracking observable behaviors of programs running on their hosts and tracking changes to file-system state. As shown in Figure 1, similar monitors run on every host used in our system, and communicate with each other for the purposes of logging software actions.

In a typical client-server interaction, an action from a client triggers an action in the server. Hence these monitors communicate with each other to precisely track the commands executed in the server in response to the actions of the client. We therefore have two broad components in a monitor. The first one that addresses isolation of processes running locally under the monitor, corresponding to *host-level* isolation. The other component is for communicating with similar monitors running on other hosts such that *network level* isolation is achieved. This is shown in Figure 2 by the division of host and network level components.

In the remainder of this section, we describe the host monitor.

The objective of our monitoring system is to identify observable events that are triggered by the execution of a program across the entire system administrative boundary. At the level of a host system, this requires us to monitor the observable actions of a set of processes. These actions are ultimately effected

through system calls, and hence, *system call interposition* is our primary monitoring approach. Each host level monitor intercepts the system calls of the applications that are running under its purview.

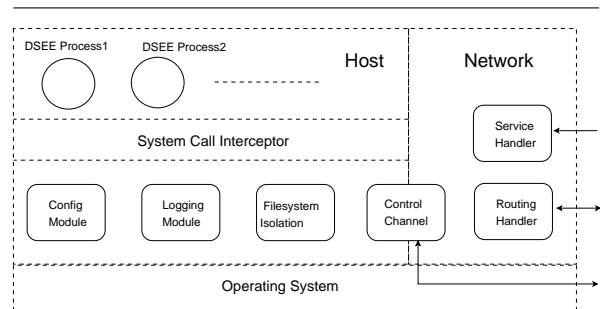


Figure 2: A host view of a DSEE.

The file system module tracks changes made by the software that is run under the DSEE. The file system module is based on our past work on one-way isolation [15, 23]. Isolation is achieved by intercepting and redirecting file modification operations made by the process running on the host so that they access a “modification cache.” This modification cache is invisible to other processes in the system. (This ensures that in the event the system administrator does not like the changes made by the software, it can be safely removed from the system without any side effects.) To ensure a consistent view of system state, the results of file read operations made by the process are modified to incorporate the contents of the modification cache. On termination of the process, the system log contains entries from the modification cache for user to inspect these files to determine if the modifications are acceptable. Otherwise, they can completely undo the changes through the trial execution.

Managing network connections When a process is being monitored, it may make connections to other hosts on the network. Once such a connection is initiated, the *Control Channel Module* (CCM) initiates the monitoring required at the other end of the connection.

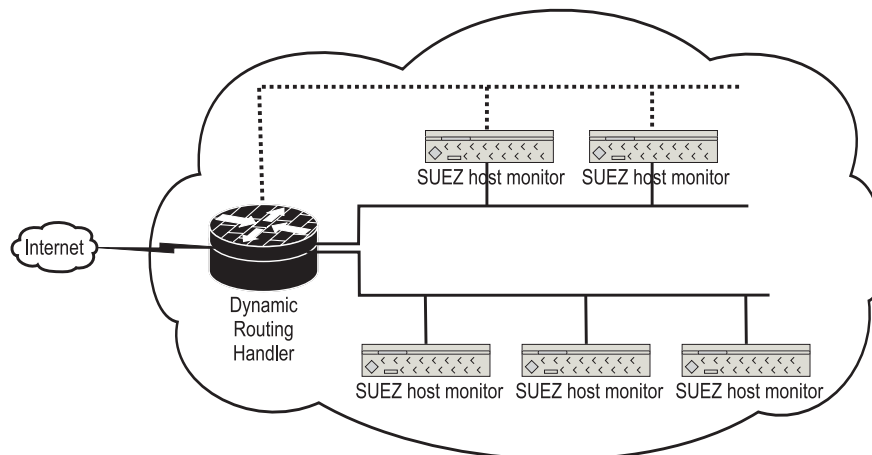


Figure 1: A network view of SUEZ.

Based on the nature of network connectivity (client/server), this module will communicate with its counterpart on the other end of the connection. If the program tries to connect to the network, CCM informs the router of this event which will result in creation of new routing path to the other hosts. There is no global network state stored at a single point for network actions since all other distributed monitors handle them co-operatively. CCM just passes appropriate control messages to the relevant components. We describe the routing module in more detail in the next section.

Dynamic service start/monitoring Recall the CUPS example, where the actions from a browser affect the configuration settings on the print server. In this case, the monitor on the remote host needs to be alerted to monitor the service that receives this request. If the service is not already running and if the SERVICE_UP message is received, then the system allows the service to be started on demand. This is accomplished by using the database of services available in the system. In case the service is already running (i.e., started through the previous step), then the monitor detects this and dynamically attaches itself to the service process. If the service is not already running, it starts the service process.

Log Module The log module generates logs depending on various configuration options and filters. These logs reside on the individual hosts. Using the system call output information itself as the log is not very useful as it may contain excessive information. The log module transforms the system call log information to a more user friendly form. Since the logs can be output can be quite long, customizable filters can be written for the logs to inspect specific actions.

For instance, the log can be customized retain information only about filesystem operations and network operations. For filesystem operations, it contains the file object name. For network operations, the service type and address related to the connection is retained. A log generator can be used to merge logs from various hosts to produce unified view logs.

Routing Module

A process that is run may connect to a network service on the local network. Isolation of this operation can be done statically or dynamically. Performing network-level isolation using static redirection requires that the system administrator knows the requirements of the software system that she is experimenting with. Guessing the requirements can soon become tedious or can impact the usability of the approach. Instead, our approach involve dynamic redirection of network service requests. Such dynamic redirection is configurable for specified network services. One question that arises in the same context is that of an application contacting an Internet host. In this case, providing complete isolation while allowing the application to run is not possible, as it is hard to emulate the functioning of an arbitrary network service. In this case, there are two options. One is to disable such requests, for the sake of security. Since our approach is built using system-call interposition, this is feasible. The other option is to only isolate the actions of the client at the host level. Of course, the disadvantage is this option is that reproduction of the entire behavior of the application is not possible, as the server side behavior is not reproduced accurately. This is acceptable as there is generally no easy solution to

```
network-op-isolation-module() {
    switch(new-route){
        case ROUTE-UP:
            client-addr = get-address-of-client();
            target_addr= get-requested-address();
            if (target-addr) already on network break
        else
            new-host = find-available-host();
            map new-host to client-addr;
            send new-routing-up message to new-host;
            get network-portion of the requested address.
            new-device = get-available-devcie();
            boot new--device.
            break;
        case DEL-ROUTE:
            client = get-address-of-host();
            find list of hosts assigned for client.
            send del-routing-path message to the host.
            new-device= get-device-name(routing-path);
            release host resources;
            release network device();
            shutdown device ();
            break;
    }
}
```

Figure 3: Algorithmic sketch of the routing module.

the problem of studying an experimental/untrusted software that tries to connect to an outside host.

Dynamically setting up routes and services requires redirection of network service requests, that are established using *dynamic route generation* and *dynamic service redirection*. We will describe the route generation in this section, and service redirection in the next subsection.

Dynamic route generation is established using a specialized *route handler* module, that dynamically establishes a routing path between the host running the program and the target host. Such dynamic redirection has several possible options – the use of forwarders that do IP masquerading such as IP tables and IP chains. However, if the application specific functionality (such as any internal tables) is dependent on the target IP address, then such forwarding mechanisms may break programs. Open source redirectors are available, however, they do not support every kind of TCP/UDP connection. Also, using a redirector requires the same to be installed on the all the target hosts. The approach we have taken is to modify the routing table dynamically on the router to forward the connections to the target network/host.

To enable redirection of connections, the host needs to configure the IP address of the target host (that runs the network service) dynamically. In our implementation, this is accomplished by establishing a virtual network interface on the target host. This virtual network interface is enabled using IP aliasing.

For a minimal set up for testing client-server implementations, our system needs one router and at least two machines, one that initiates a service request and the other that accepts such requests. (These can be set up in an inexpensive fashion using virtual machines, a topic we will discuss below.) If each of the machines needs to be on a different subnet, then the router should have a network interface on each subnet. Furthermore, IP forwarding needs to be enabled in the kernel state of the router. Our router module is required to be running on the router and on the host accepting service requests. This is needed to change routing tables dynamically.

Let us look at a typical client-server interaction between a client and a web server on our system.

1. A client invokes connection request to the service that either runs on the network or is not yet available.
2. The Control channel module on the client intercepts this event and notifies the routing module (running on the router) of the address for this connection request.
3. Upon receiving this request, the router checks whether there exists an already running web server on the network. If so, it returns and the CCM informs this service-related information to a the service handler on a machine running server. If the service is already running on the server, the client can start exchanging messages. If not, the service handler starts the service. If the network path is not established it proceeds to the next step.
4. The routing module on the machine receiving the message from the router boots up a new virtual network interface with the address.
5. The router chooses appropriate address for a new routing path and boots this new interface.
6. From this point onwards, all communication is transparently redirected through this newly established path between client and server.

The routing module is explained in Figure 3. The state maintained in the router consists of available Ethernet devices and addresses of hosts running. During the initialization of the router module, devices' name need to be given to the module as parameters. When a new host comes up on the network, it registers its address with the router module. The router module maintains a vector of such addresses. Whenever a task is complete, the network interfaces allocated for the routing path are brought down, and the returned to the pool of resources for future use.

Message Handler

Often, the focus of attention on a particular trial execution is in executing one or more features of an application. In this case, a user may want to only focus

```

message-loop() {
    while(true) {
        waitfor-command();
        dispatch-command();
    }
}

dispatch-command {
    case NEW-ROUTE-UP:
        /* set up new route */
        break;
    case DEL-ROUTE:
        /* delete route and release resources */
        break;
    case SERVICE-UP:
        /* bring up the network service */
        break;
    case SERVICE-DOWN:
        /* shutdown network service */
        break;
    case NEW-HOST-UP:
        /* add host info to host list */
        break;
    case QUERY-HOST:
        Query host list ;
        break;
    case START-TRACING:
        /* start recording operations */
        break;
    case STOP-TRACING:
        /* stop recording operations */
        break;
}

```

Figure 4: Various messages received by the message handler.

on this operation and ignore other operations of the system. A message handler is made available on each client to start and stop tracing the operations made by the trial execution. A typical use scenario is as follows: When the user would like to focus on exercising a feature in the application, before exercising this feature, she can instruct the client DSEE to send a `START_TRACING` message. All the DSEEs will record the subsequent operations made by the task. After the user is done, she can send a `STOP_TRACING` message that will stop recording the operations of the task. When tracing is stopped, the set of actions that were recorded between the `START_TRACING` and `STOP_TRACING` messages capture observable effects of the operations in this window.

Additionally, the message handler also deals with messages from other DSEE components. These messages are about routing information and services registration. On receiving these messages, the message handler invokes the appropriate handlers. The responses to messages received are shown in the commands exercised by the message handler in Figure 4. For example, when it gets `NEW-ROUTE-UP` or `DEL-ROUTE` messages, it invokes routing module to boot up or shutdown routing paths respectively.

If the application running in the DSEE is untrusted, it may send false messages to the message handlers on the other hosts. For this purpose, the default policy enforced by the system call interceptor is to disallow any such messages on the control channel that is maintained by the host monitors.

Support for virtual machine hosts Virtual machines can result in creation of inexpensive hosts on demand, and our approach is designed to take advantage of the use of virtual machines. Our prototype implementation uses VMware virtual machines [5], where creation/loading of virtual network interface and virtual network groups can be easily done on demand.

Experimental Evaluation

Before describing the experiments performed with SUEZ, we describe our experimental set up. We also describe the configuration options available to the user.

Setup

- *Virtual network setup* The network set up has one router and two subnets. Since we used VMware to create hosts on the network, this required creation of three virtual machines.
- *Router setup* To act as a router, the kernel value for `IP_FORWARD` should be 1. This router has three network interfaces, one on the physical network, and the other two for subnets A (192.168.1.X) and B (192.168.2.X).
- *Message handler setup* The message handler on the router is set up with available device names and addresses. Above case, available device on subnet B is bound to 192.168.2.1.

- *Server Host monitor setup* A SUEZ host monitor (with its associated message handler) is launched on a machine on subnet B to act as host available for service. To this, `HOSTMODE` value need to be set in the config file. At the starting of this host information will be sent to the message handler.
- *Client Host setup* A SUEZ monitor with `ROUTEMODE` value set in the appropriate config file for the a machine on subnet A. `ROUTEMODE` config variable is explained below.

From this point onwards, if the client program tries to connect to a service, with SUEZ with `ROUTEMODE` set, the connection will be transparently forwarded to host in subnet B.

Configuration Parameters

The following configuration flags need to be set on the hosts in the network.

1. `ROUTEMODE` – If this value is set, SUEZ will intercept all network connections before the client program get connected to its original destination. Eventually, the connection will be transparently forwarded to a machine that hosts the corresponding service.
2. `HOSTMODE` – To automatically configure an ip address and start the required service dynamically as on host, one would set this value in SUEZ. If this flag is set, the host monitor in SUEZ will send host address to the appropriate message handler.
3. `REMOTELOG(ULOG)` – In order to make a unified log, each host monitor traces and collects the events of interest. If this value is set in config file, each event of interest will be logged. When these events are merged into to one log, only events of interest will be made viewable in the unified log.

In addition, a list of available devices available to setup new routing paths on the router is provided as input to the router module through a config file.

In the following section, we present an experimental evaluation of using our approach. Our evaluation consists of two parts: the first is a *system evaluation*, which was about applying the system to study the execution of several system and application software tools. The second part is a performance evaluation of our system.

We analyzed the installation and execution of several applications in DSEEs created using our system. Below we describe four candidates from our experiments.

Address Book leak in SquirrelMail Squirrelmail is a Mail User Agent (MUA) package written in PHP4. Being a web based user agent, it interacts with a web-server in addition to a mail server. The functionality of the program is triggered through many links and buttons

on the web page interface. For SquirrelMail, since the interface is web-based, we tried to understand the functionality that interacts solely with the web server, as opposed to that which also interacts with the mail server. Understanding the nature of information stored in a web server is critical as the protection of data stored in a web server is an important issue. So we installed Squirrelmail with its default configuration in a DSEE, and observed the actions during installation.

After the installation, we tested the various options in Squirrelmail by trying out the various options in the web interface. One such interface is the address book interface that allows a user to add or remove entries from his address book. Once that interface was tried, the results of the system pointed to file modifications on the web server. We observed that the default configuration resulted in placing the data sub-directory that holds the address book information under the top-level Squirrelmail directory. If this URL is known, an arbitrary user can access the (private) address book information of any other user. The URL is normally known to any user of the system, and is easily guessable if one knows the presence of a Squirrelmail installation on a server. This directory needs to be protected from being directly accessible in order to protect the privacy address book information.

Our tool enabled us to correlate the action of creating an address book entry on the client to the location that it was stored in the server and therefore uncover the vulnerability of address book information leak. Changing the access permissions for the directory subsequently solved this problem.

Remote web server upgrading Several systems exist that perform upgrades/installation from a remote machine. For instance, Webmin [9] is one such tool. The primary purpose of such tools is to simplify desktop administration. Although this purpose is achieved by such tools, they do not provide a way to recover from any problems during installation. For instance, if the installation of a package using a remote administration tool is not successful it is difficult to recover from such errors. Configuration files that are overwritten may be lost during the installation process. (Using a backup procedure, the system administrator can save these files, but this requires knowing in advance which files are being overwritten). Using our approach, we can perform the installation without the fear of damaging the system in any way, and then finally inspect the system to see if the changes made by the installation are desirable, and then commit these changes. If the installation does not proceed as expected, they can go back to the original state of the system.

To study the use of the Webmin administration tool, we upgraded the apache web server program from a remote client machine (in a DSEE). We upgraded the http package (that contains the apache web server) version 2.0.55 to http-2.0.58 through the

Webmin tool remotely. After the installation, we tested whether the installation process worked fine by testing the new version of the web server. Also, we observed for any modifications to existing files. In both cases, there were no problems resulting from the installation. Hence these changes were successfully committed into our system.

Debugging Mgboard Configuration Mgboard [6] is web-based message board on apache with php. Mgboard uses an internal flat-file database rather than an external SQL database such as MySQL. Using Mgboard, a user can post articles and upload files to the website. In interactive programs such as Mgboard, it would be a tedious task to identify misconfigurations. For example, the files that store system configuration data for Mgboard needs to be group-writable and other when create database file (for public writing). As server-side scripts are hard to debug, any misconfigurations in Mgboard (which is indirectly powered by the Phpadmin program) are hard to detect. Also it is difficult to know which actions (such as addition/update of posts) are affected by this misconfiguration. However with the use of a DSEE, it is easy to know which actions were triggered (specifically, which files were executed) and thereby reason through the CGI script operations.

To check this, we performed two experiments. In the first experiment, we intentionally planted certain misconfigurations in the remote client, by introducing file permission errors. In the second experiment, using a web client, we created a new page on the web server. While posting an article from the client on to the server, using our tool, we observed creation of temporary files in the /tmp directory of the server. This helped us to investigate the possibility of a local race condition vulnerability that could result through the creation of this temporary file. Such a race condition could happen if arbitrary users can overwrite this file. Such reasoning is possible with our system as the unified log presents the temporal sequence of such actions and writing custom filters can identify and 'zoom in' on the error.

Configuration errors in Proftpd ProFTPD [3] is a ftp server written for use on a UNIX-like operating system. A ftp server allows the remote user to perform operations on remote file systems, and even send site-specific commands. It is therefore important to test the software installation and check for the potential actions of the server when it interacts with a client. Occasionally the settings may dictate account users or even anonymous users can inquire or change file systems explicitly by using remotely issued commands.

We tried this as a candidate example for testing the ProFTPD server. We observed that on installation, the system executes an init script, which results in two main actions: creation of user and group ids for the server. Another configuration file that was created was the /usr/local/etc/proftpd.conf. Also during our installation, the server was configured to restrict users' access

with DefaultRoot in proftpd.conf but an accidental system configuration error resulted in the option #DefaultRoot . When the service was started, we exercised the gftp client to change the working directory to the root directory and store a file. The usual expectation on part of the server administrator was to have the file stored in user home root, but due to this misconfiguration it triggered a permission error. In this case, the unified view log shows actual action sequence with operations on the file system starting from operations on the client to the server. It pinpointed the source of the error to the DefaultRoot option. Such configuration errors can be debugged effectively with our approach, which allows one to focus specifically on the results of a particular action in a program.

Performance Evaluation

The second part of our evaluation is the of the performance of our system with several client-server applications.

We describe the experimental setup first. We used several virtual machines hosted in machine for all our experiments. The machine is an AMD Sempron 2400+ CPU with 2 GBytes memory, running the Red Hat Enterprise Linux distribution. The virtual machines also run the RedHat Enterprise 4. Each virtual machine instance runs SUEZ with router and service handler, and any associated client or server programs.

We classify the performance measurement experiments into three categories:

- *Client-side overheads.* These overheads in the client side may result from the monitoring overhead through SUEZ.
- *Server side overheads.* These overheads result from the monitoring overhead at the server.
- *Network delays.* These are overheads introduced due to the routing and service isolation in SUEZ.

- *Service and program launching overheads.* Since the service redirection and service program startup are done on demand, this may introduce additional overheads during the start of a session.

We discuss all the overheads in detail below.

Client-side overheads We recall that our system uses system call interposition at the client to track the actions of the client, and any possible communication messages to the server. Such interception facilities are implemented using *ptrace* mechanisms available in the Linux operating system. We have measured the performance as a ratio of the combined system and user time and compared them for the following situations: a) without any monitoring b) with the use of our monitoring mechanism. Figure 5 shows the performance overheads at the client. The performance numbers show overall execution times with and without the monitor.

In the figure, we have measured the performance of four desktop clients while performing the experiments mentioned in the previous section. The results show that the overhead due to system call interposition vary for various clients ranging from 68 to 325%. The difference in overhead is due to the frequency of system calls invoked by these different clients. In addition, an entirely a user-level mechanism such as *ptrace* suffers from moderate to high overheads [15]. These high overheads due to the context switching associated with the process that performs the monitoring. A kernel level mechanisms typically has overheads in the range of 10-15% as evidenced by earlier work on kernel level mechanisms for one-way isolation [23].

Server-side overheads For servers, we measure the overheads differently. Since most servers continue to run even after servicing client requests, it is not possible to measure the overheads in a manner similar to that of

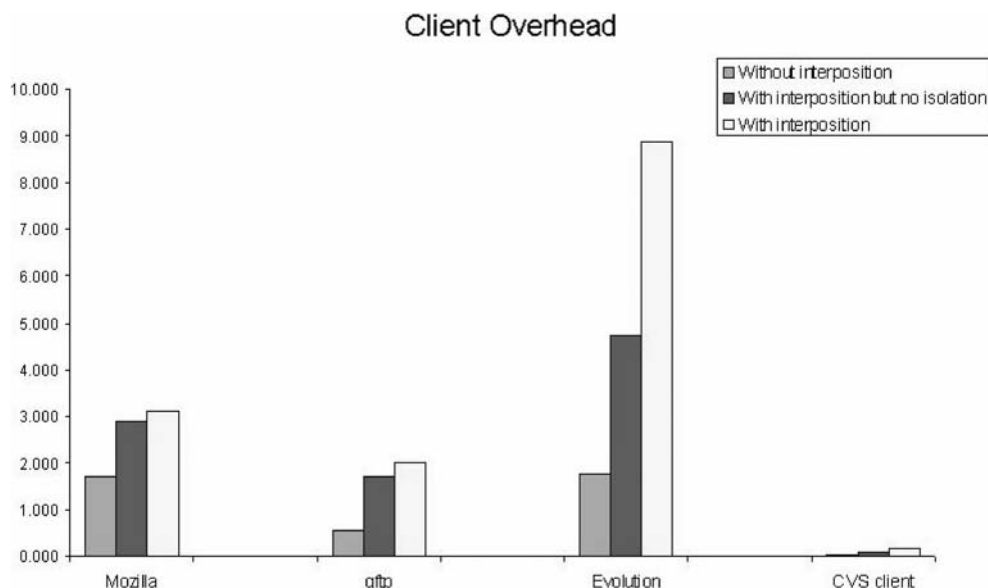


Figure 5: Client-side overheads.

the clients. To measure the overheads on the server, we have measured the mean response time of the server at each client. Recall that our system monitors the system calls made from the client, and on a connect system call, sets up routing paths and starts the corresponding services. Therefore, the response time is measured (at the client) as the difference from first connect system call to first or last recv call on each client's log.

Figure 6 shows the mean response times at the client. This response time indicates the steady state overhead, i.e., the overhead without the following two causes 1) any (one-time) routing overhead in the computation of the virtual routing path and 2) overheads from automatic start of the corresponding network service. As shown in the figure, the response times for various server programs are within a factor of two. For trial installation purposes we consider such overheads acceptable. Moreover, a kernel level patch to the isolation mechanism will reduce the response time overhead.

Route Computation Overhead

In order to create a dynamic routing path, new network interfaces are needed to be initialized on the router and the server host respectively. Delays are introduced at the router (in the control channel module implementation) to find any available hosts for assignment of new IP address.

We compute the overheads for CUPS, Webmin, Proftpd, Sendmail and CVS. overhead also as the difference between mean server response time at the client, with and without route computation. In order to measure this overhead introduced by the dynamic route computation and the service handler, we obtained the following measurements of the programs used in our experiment. These are a) the relative mean response time without the system, b) with the use of isolation but not using routing and service function on servers and c) with host-level isolation and the use of dynamic routing and service handling. The performance for all the seven

applications that were used in the server side overhead experiments (described above) were measured. The time stamps on the client were recorded in the log for each network and file related operation. For web based programs, the mean time difference from first connect to first recv system call was measured. For sendmail, the difference between the first connect to last socket write was measured. For proftpd the difference between the first and second connect system calls made (the first call is made for getting the data channel for the file transfer). For CVS, the mean time to create the .cvspass was measured.

We measured the delays introduced due to the routing process. This delay does not depend on the specific application that was used. We measured this delay as an average delay as perceived by the client. The mean delay introduced due to the router (as perceived by the client) was measured to be 0.125 seconds.

Routing and Service Launching Overhead

Once a host receives a request for a service, it needs to start the service and subsequently the server responds to the request. We measured the service launching time for each of the server applications tested. We measured this as an average delay perceived by each client. Also, to avoid routing delays from entering the picture, we set static routes from the client to the server. These are the delays for the server applications: sendmail (3.8 s), apache (3.8 s), Proftpd (2.2 s), and Webmin (1.6 s).

We note that services can be started using `inetd`, and may not result in overheads when the client contacts the server host for the first time. Such service programs can be traced dynamically (i.e., attached to the monitoring process). This will result in much lesser overheads. We also note that at the time of interception of the original network operation, the route is created and the service is launched transparently before the actual connection request from the client is sent.

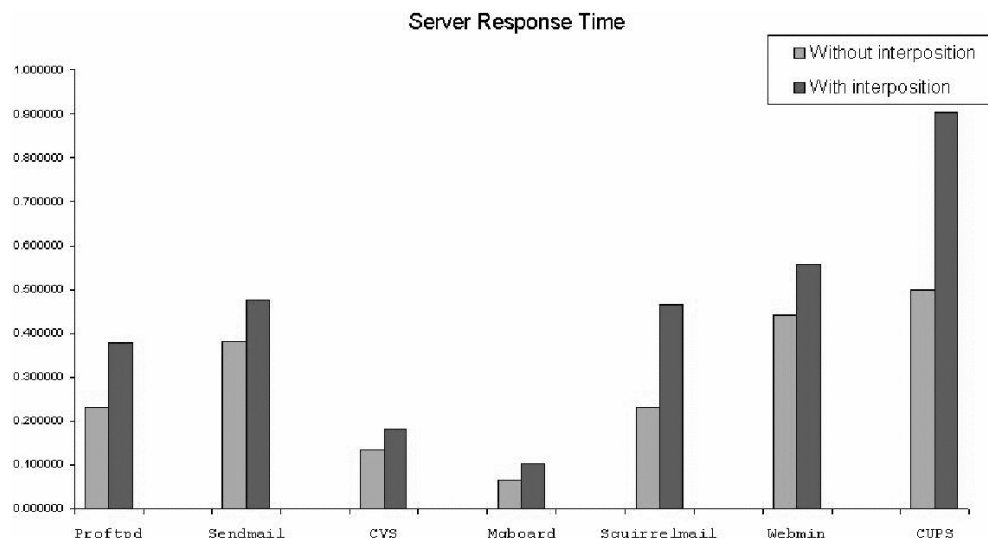


Figure 6: Server response times.

Conclusion

A set of questions that the system administrator typically has when performing any trial execution are:

- During the trial experiment, does this piece of software cause conflicts with other packages such as overwriting configuration files?
- After installation of a package, does it work cooperatively well with existing software within the entire network?
- Does any of the features of a piece of software malfunction, even though it may seem to work well apparently?
- Is it safely deployable in the network? Does it violate the network privacy and security policies? Does it install files in hidden locations?

The system we describe in this paper, called SUEZ, is designed to support assist a system administrator in answering these questions. To achieve this our system employs one-way isolation of local and remote operations inside a distributed safe execution environment. In addition to satisfying main goal of providing support for study and experimentation with software, our approach has numerous other benefits. It requires no access to source code of the applications that need to be studied; it is cost-effective in being able to utilize virtual machine technology for dynamically configuring hosts and routes, and customizable to various situations that one may encounter in system administration practice. We believe that our approach has the potential to be applicable in several day to operations involving system trials, reverse engineering and troubleshooting.

Acknowledgments

We thank Zhenkai Liang, Weiqin Sun and R. Sekar for many discussions about the implementation of distributed isolation operations that formed the basis for writing this paper. We also thank our shepherd Narayan Desai and the anonymous referees for reading our text and providing many useful suggestions that have improved the contents this paper. Finally, we acknowledge Rob Kolstad's help with typesetting of the manuscript.

Author Biographies

Doo San Sim is a graduate student in Computer Science at University of Illinois at Chicago. His research interests are in computer security, mainly in addressing security in software installations. He can be reached by email at dsim2@uic.edu.

Dr. V. N. Venkatakrishnan is an Assistant Professor of Computer Science at the University of Illinois at Chicago. He is currently co-director of the Center for Research and Instruction in Technologies for Electronic Security (rites.uic.edu). His main research area is computer and network security. Specific research areas include malware detection, software security and

personal information privacy. He is available by email at venkat@cs.uic.edu.

Bibliography

- [1] *Common UNIX printing system*, <http://www.cups.org>.
- [2] *Controls the system log*, Man pages.
- [3] *Professional FTP*, <http://www.proftpd.org>.
- [4] *Strace*, <http://www.liacs.nl/~wichert/strace>.
- [5] *Vmware*, <http://www.vmware.com>.
- [6] *A web board not using Sqldb*, <http://www.php.school.com>.
- [7] Acharya, A., and M. Raje, "Mapbox: Using parameterized behavior classes to confine applications," *USENIX Security Symposium*, 2000.
- [8] Brown, A. and D. Patterson, "Undo for operators: Building an undoable e-mail store," *USENIX Annual Technical Conference*, 2003.
- [9] Cameron, J., *A web-based interface for system administration for UNIX*, <http://www.webmin.com>.
- [10] Cespedes, J., *A library call tracer*, Man pages.
- [11] Chen, P. M. and B. D. Nobl, "When virtual is better than real," *Proceedings of Workshop on Hot Topics in Operating Systems*, 2001.
- [12] Chiueh, T., H. Sankaran, and A. Neogi, "Spout: A transparent distributed execution engine for java applets," *International Conference on Distributed Computing Systems (ICDCS)*, 2000.
- [13] Dan, A., A. Mohindra, R. Ramaswami, and D. Sitaram, 'Chakravyuha: A sandbox operating system for the controlled execution of alien code, Technical report, IBM T.J. Watson research center, 1997.
- [14] Goldberg, I., D. Wagner, R. Thomas, and E. A. Brewer, "A secure environment for untrusted helper applications: confining the wily hacker," *USENIX Security Symposium*, 1996.
- [15] Liang, Z., V. Venkatakrishnan, and R. Sekar, "Isolated program execution: An application transparent approach for execution of untrusted programs," *ACSA Computer Applications Security Conference (ACSAC)*, Las Vegas, December, 2003.
- [16] Malkhi, D. and M. K. Reiter, "Secure execution of java applets using a remote playground," *Software Engineering*, Vol. 26, Num. 12, 2000.
- [17] Muniswamy-Reddy, K.-K., C. P. Wright, A. P. Himmer, and E. Zadok, "A versatile and user-oriented versioning file system," *Proceedings of USENIX Conference on File and Storage Technologies*, 2004.
- [18] Prevelakis, V. and D. Spinellis, "Sandboxing applications," *Proceedings of Usenix Annual Technical Conference: FREENIX Track*, 2001.
- [19] Provos, N., "Improving host security with system call policies," 2002.

- [20] *Recovery-oriented computing*, <http://roc.cs.berkeley.edu>.
- [21] Santry, D. J., M. J. Feeley, N. C. Hutchinson, and A. C. Veitch, "Elephant: The file system that never forgets," *Proceedings of Workshop on Hot Topics in Operating Systems*, 1999.
- [22] Scott, K. and J. Davidson, "Safe virtual execution using software dynamic translation," *Proceedings of Annual Computer Security Applications Conference*, 2002.
- [23] Sun, W., Z. Liang, V. N. Venkatakrisnan, and R. Sekar, "One-way isolation: An effective approach for realizing safe execution environments," *NDSS*, 2005.
- [24] Whitaker, A., M. Shaw, and S. Gribble, "Denali: Lightweight virtual machines for distributed and networked applications," *Proceedings of USENIX Annual Technical Conference*, 2002.
- [25] Yu, Y., F. Guo, S. Nanda, L. Lam, and T. Chiueh, "A feather-weight virtual machine for windows applications," *Proceedings of the 2nd ACM/USENIX Conference on Virtual Execution Environments (VEE'06)*, 2006.

WinResMon: A Tool for Discovering Software Dependencies, Configuration and Requirements in Microsoft Windows

Rajiv Ramnath – National University of Singapore

Sufatrio – Temasek Laboratories, National University of Singapore

Roland H. C. Yap and Wu Yongzheng – National University of Singapore

ABSTRACT

This paper describes WinResMon, a system tool for determining resource usage and interactions among programs in Microsoft Windows environments. Think of WinResMon as a debugging tool to assist with software maintenance in a Microsoft Windows environment. It shows the current system state in terms of how resources are used and explains how the system arrived at that state. WinResMon can be used to determine how a program uses the registry and which files are needed by that program. WinResMon differs from other systems/tools in that it is integrated, designed to answer queries about resource usage and dependencies over time, and extensible, allowing the addition of new functions and tools.

Introduction

The tasks of software maintenance and configuration require a precise understanding of system resources, the individual requirements of each piece of software, and interdependencies between every software program on the system. We use the term *software maintenance* to describe the system administration task of ensuring that software on a system is configured and maintained correctly over time. Examples of software dependencies include:

- file sharing: including dynamic libraries and external data storage
- sharing of software configurations: usually in the form of registry keys in Microsoft Windows
- interprocess communication and synchronization.

Although software maintenance tasks might seem conceptually trivial, they can be time consuming and difficult, especially in large or complex environments. System administrators often rely on documentation and on-line information such as FAQs or forums, but such information is often incomplete.

In various distributions of Linux, software dependency issues are partly addressed by the use of a package manager. The Red Hat Package Manager (RPM) [1], for example, records the currently installed packages and the files required and provided by these packages in a centralized database. RPM checks dependencies before removing a package to ensure that files required by other installed packages are not removed. Similar checks are done to prevent installing a package which contains files that conflicts with other existing packages.

In Microsoft Windows, however, software installation can be complex, and the exact dependencies between different software programs might not be

clear. The confusion is further compounded by many implicit software interdependencies, e.g., registry keys which are part of shared software configurations. As a result, it is difficult to know whether to remove a file when uninstalling an application. File removal might lead to problems with another piece software or might create security vulnerabilities.

When installing two or more programs that share files, one program may cease to function correctly because the installation of the second blindly overwrote shared libraries (DLL files). There is also the question of when to perform a major software upgrade. System administrators may delay upgrading for fear of breaking existing software; yet such a choice has its own risks.

This paper focuses on the problem of software maintenance in Microsoft Windows NT-based operating systems (Microsoft Windows XP, Microsoft Windows 2000, Microsoft Windows 2003).¹ We present *WinResMon*, a discovery and system debugging tool for determining a program's resource usage as well as the resource usage interactions between multiple programs.

Motivation and Applications

We believe that the key to solving the software maintenance problem is to understand the life cycle of the system and programs therein. We also wish to empower ordinary users, removing the requirement of knowing every minutiae of Microsoft Windows. Although WinResMon is not tailored specially for system security, it can also be utilized as a security auditing tool.

We designed WinResMon to act as both an infrastructure or framework and a system utility. As a

¹While an appropriate version of a tool similar to WinResMon could also be of use in UNIX, its value is much greater in a Microsoft Windows environment.

framework, it is extensible, and one can therefore add new functionality and build customized tools. As a tool, it comes with pre-built modules to answer typical questions about resource usage and dependencies.

When used as a debugger, WinResMon investigates the current system state, i.e., which program uses which registry keys, and determines how the system has arrived at that state. WinResMon accomplishes this by recording information about the evolution of system software dependencies and resource usage over time. To solve general problems in software maintenance, WinResMon monitors: files, the registry, and interprocess communication and synchronization. However, it is not feasible to continuously and permanently record all changes to the system since the required space would be prohibitive. WinResMon employs a reasonable compromise by maintaining detailed usage records over the current time period and a subset of information that can be maintained over the lifetime of all software in the system.

We illustrate the software maintenance problem with some simple examples. One attack vector for spyware is to register itself as a start-up program, thereby hiding itself from the end user. Also consider a music player which may require some sound decoding libraries. This application only functions correctly with certain versions of the libraries. Thus, replacing a library can lead to software failure. Various pieces of software may also conflict, e.g., two mail transfer agents (MTAs) usually do not co-exist.

Some common system administration questions and tasks which WinResMon can assist with include:

1. ***Can we safely remove a particular DLL file?***

Some applications provide shared libraries (DLL files) for use with other applications. When the system administrator uninstalls an application, she can also choose to remove the DLLs. Removing a DLL can cause other programs which still use it to malfunction. On the other hand, blindly retaining all DLLs will cause the system to keep growing and may create security vulnerabilities. The system administrator generally lacks adequate information to determine whether another program uses a shared library. WinResMon can be used to record the utilization of each DLL so that the system administrator can determine which programs use which DLLs.

2. ***Why does a program need administrator privilege to run?***

Running programs with administrator privilege is discouraged because malware such as viruses/spyware or poorly written applications can damage the system. However, some programs may need to run as the administrator without an obvious reason. WinResMon can detect whether a program needs administrator access. The idea is to understand the reasons for

elevated privileges and configure the system to limit the use of privileges. If it finds applications that require certain administrator privileges to function correctly, the system administrator can set up a policy that restricts the administrator privileges to the needed resources (files, registry keys, etc.). We remark that this approach can be contrasted with confinement systems such as systrace [2] in UNIX. WinResMon is an auditing tool, it does not confine system calls, but provides useful input for system administrators to create policies on resource access which can then be used to limit privileges.

3. ***Monitoring sensitive registry locations to detect spyware.***

Managing the Microsoft Windows registry is difficult due to its complexity. Spyware often takes advantage of this complexity to bury itself in the registry, making it difficult for the user to remove it completely. In [3], the authors have listed the most common entry points for spyware to enter a Windows NT system. The following are some of the configuration settings WinResMon can monitor:

- *Autostarts*: monitor which programs load on startup.
- *Internet explorer hooks*: track hooks which define the default search page, toolbars and browser helper objects (BHO), etc.
- *Winlogon*: look for applications that hook into system resources.
- *Services*: monitor services such as automatic startup services (e.g., task scheduler) or drivers which are installed as services.
- *DLL injection*: monitor DLL injection attacks (any application that uses user32.dll can be hijacked by having a DLL injected into its process space).
- *File associations*: monitor the registration of file extensions with applications. For example, .DOC is registered to Microsoft Word.

System Design

The WinResMon system infrastructure shown in Figure 1 consists of the following components: logger, archiver, query API, and user-log API. The logger generates resource-access traces which are later used by the analyzer. The archiver performs log compaction/summarization of old traces. Query and user-log API provide the interface to the trace database.

The Logger

The logger consists of a system call (syscall) interceptor and a trace generator module. The syscall interceptor is an in-kernel driver which monitors system calls made by each process and sends the monitored event information to the trace generator. The trace generator is a user-space service (daemon) which collects event information sent from the syscall interceptor and

generates resource access traces. Ideally, the logger is meant to run *all the time* so as to record the entire life cycle of how resources are used by software.²

A log trace file consists of a list of records, each representing an access operation on one of the following types of resource:

1. *File*: covering both directories and regular files
2. *Registry*
3. *Process*: mainly to record process creation
4. *Synchronization objects*: which records information on inter-process synchronization mechanisms provided by Microsoft Windows (mutex, semaphore, event and waitable timer)
5. *IPC*: including named pipes and mailslots.

In addition to these five basic resource types, WinResMon also captures *system event* information such as: process termination, system boot and shutdown, user login and logout. Moreover, it also provides a user-log API for users or applications to insert user/application-defined milestone events. One potential usage of custom events is to demarcate and distinguish the software installation and uninstallation portions of the log.

A record entry contains common information and specific information relevant to that record type. The common information consists of: *record-type*, *time*, *process-id*, and the *error-code* (in the case of failure). The specific information recorded by different record types are:

1. **File:**
 - absolute path of the file
 - file operation: *open*, *read*, *write*, *delete* or *move*
 - operation-specific information. For *open*: the access flags. For *read* and *write*: the number of bytes read or written. For *move*: the new path.
2. **Registry:**
 - absolute path of the registry key
 - operation: *open-key*, *query-value*, *set-value*, or *delete-key*

²One might only run the logger at select times instead, but this means WinResMon could miss critical information.

- operation specific information. For *open-key*: the access flags. For *query-value* and *set-value*: the type, size, and value of the registry key.

Due to the importance of the registry in software maintenance, WinResMon logs the actual data changes made to the registry; whereas for file I/O, it is not practical to record the data.

3. Process:

- absolute path of the executable corresponding to the newly created process
- command line arguments
- process id of the newly created process.

4. Synchronization objects:

- type of the object: *mutex*, *semaphore*, *event*, or *waitable-timer*
- name of the object
- operation: *create*, *open*, or *delete*.

We are interested only in objects which have names in the system-wide namespace, because anonymous and process-wide named objects will not interfere with other programs, thus they are unrelated to software dependencies.

5. IPC:

- type of the IPC: *named-pipe*, or *mailslot*
- name of the IPC
- operation: *create*, *open*, *delete*, *send*, or *receive*.

WinResMon records the synchronization objects and IPC operations listed above since they involve global (system-wide) namespace which could be used by different programs to interact with each other. One can use WinResMon to uncover the causes of the following problems:

- a) If process *A* has created a semaphore *s*, and process *B*, which is unaware of the existence of *A*, is trying to create a semaphore with the same name *s*, *B*'s operation will fail.
- b) If process *A* fails to run correctly and semaphore *s* is not created, then process *B*, assuming the existence of *A*, will fail trying to use *s*.

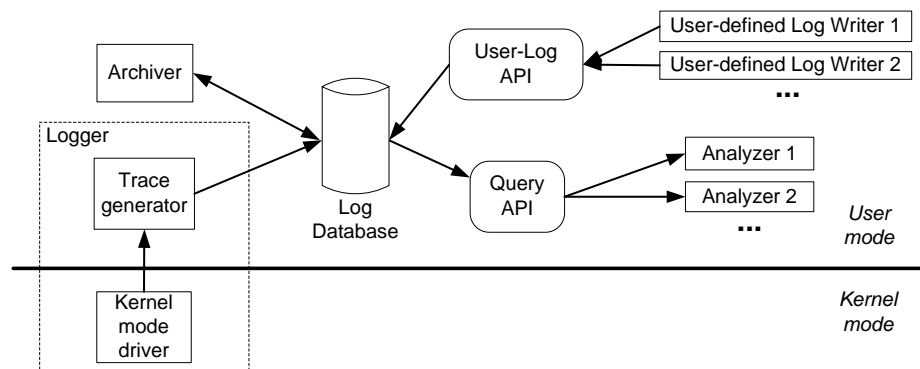


Figure 1: WinResMon overall system architecture.

6. System event:

- system event type: *process-termination*, *boot*, *shutdown*, *user-login*, or *logout*
- any event specific information: path of the executable of the process in the case of process termination, user name in the case of user login/logout, etc.

7. User defined event:

- the path of the executable of the process generating the event
- a binary string describing the event.

Not all operations need to be logged into the database as some resources are not significant to record, e.g., the temporary directory, C:\temp. A filter can therefore be used in the logger to prevent logging resource access of no interest to the system administrator.

The Log Database

The log database consists of a number of log files and one distinguished *active log* to which the logger records current resource access activity. To maintain reasonable log file sizes, WinResMon performs a “log switch” to create a new active log for recording subsequent entries. The old log files are then subject to the log compaction/summarization process by the archiver. Any of the following conditions can trigger a log switch:

- The size of the current log file reaches the specified *Max_log_size*.
- The number of entries in the current log file reaches *Max_log_entries*.
- The user manually initiates a log switch process.

Archiving Old Traces

As WinResMon logs system activities over a long period of time, the trace becomes very large. If old traces are discarded, some early yet potentially valuable information, such as identifying which program first created a file, is lost. Since it is necessary to eventually prune some information to avoid excessively large log files, WinResMon summarizes old trace files into a “*compacted trace database*”. This maintains a balance between compactness and the ability to answer important questions about system resource access.

There are two main issues in performing trace compaction, determining when to initiate the compaction and how to perform compaction on old trace entries. WinResMon uses a module called the archiver which runs based on a specified *Archive_interval_check* time interval. Upon activation, the archiver compacts previously uncompact entries which are older than the specified *Old_log_age*. WinResMon employs the following strategies in performing the compaction:

Log Entry Summarization/aggregation

WinResMon can summarize multiple entries with similar information to produce an aggregated entry in the compacted trace database. It applies the following policies on various resource types:

- **File:** multiple read or write operations on the same file are summarized by recording the time of the first and last operations and the total number of time the operations were done.
- **Registry:** multiple query-value operations on the same registry key are aggregated. Multiple set-value on a key are aggregated only if the values written are identical.
- **IPC:** send and receive operations of one IPC object are aggregated by recording the time of the first and last operations.

When matching a query on an aggregated entry which records a time interval, WinResMon considers that the entry satisfies the specified time constraint if *one* of the values matches the constraint.

Selective Priority-based Entry Removal

One strategy to reduce the old traces involves removing entries deemed to be of little value for answering future questions. WinResMon implements selective entry removal strategy based on a user-supplied configuration file. The configuration file assigns a priority to log entries. For example, a log entry for writing a registry key might be considered more important to keep than one for reading a registry key.

A fragment of an example configuration is shown in Appendix 2. This example uses priority values ranging from 1 (least important) to 5 (most important). For each resource type, configuration entries are matched in a sequential order, and mappings are listed from most to least specific. It is possible to omit some arguments, i.e., with wildcards. To simplify the priority assignment, WinResMon classifies file open operations in Microsoft Windows into one of three modes: RO (read-only), RW (read-write) and WO (write-only/append). The archiver translates the semantics of each operation from file flags in the raw log entries to the appropriate values for matching against the configuration.

The existing applications and anticipated usage, together with some general principles, can be used to derive the priority assignment for the configuration. One general principle is that “transient information,” such as that those on synchronization objects and IPC, becomes less relevant after system shutdown and can be given low priority. During the removal process, all log entries with priority lower than the *Lowest_priority_retained* value will be purged.

Auto Deletion of Old Log Files

To maintain reasonable storage usage, WinResMon eventually needs to remove entries deemed too old. The log file whose newest entry timestamp exceeds *Max_log_lifetime* is deleted. If necessary, it is also possible to additionally provide an API function (in a secure manner) to perform deletion on selected trace entries based on a specified selection condition.

A Query API and Analyzer

We want to support trace analyzers which answer queries solving particular software maintenance problems.

WinResMon therefore provides a query API which can be used to obtain information from the trace database. One can view the trace as a database and the provided query API as the query language by which the analyzers extract relevant information from the database.

The main query API, analogous to an SQL Select command, is `trace_select([selection condition], [field projection], [time interval], [output order])`, where the caller specifies the matching condition(s) and fields to return. The *selection condition*, specified on string types such as program name and registry key path, takes the form of a regular expression. Logical operators can be used to combine matching conditions.

Continuing the database analogy, the caller uses projection to specify which fields to return. For example, suppose one only wants to know the list of programs accessing a certain file. In other cases, one wishes to know all access operations on the file (i.e., the order and access time matter). In the former, the analyzer only returns a set of program names. In the latter, it returns the sequence of all access operations.

We specify *time* in the format below:

1. "YYYY-MM-DD hh:mm:ss": an explicit timestamp
2. "-[count][m|h|d]": a relative time earlier than the current time
3. "OLDEST": the time of the oldest available entry in the log
4. "NOW": the current time.

The time interval is then specified as: "[start time] TO [end time]".³ Some examples of time interval definitions are:

- "2006-01-25 11:47:51 TO 2006-08-21 13:41:16"
- "-1d TO NOW" (in the last 24 hours)
- "OLDEST TO -8h" (everything except the past 8 hours).

Output order controls the ordering of query results. It can be either: FORWARD, to list the oldest entry first or BACKWARD, to show the most recent entry first. The BACKWARD option is useful to get the *k* most recent operations.

After obtaining the *trace_handle* from `trace_select()`, one can call `trace_next(trace_handle)` to retrieve the records and `trace_close(trace_handle)` to finish retrieval. Some sample analyzer applications using this query API are described later.

The User-Log API

The user-log API lets applications add their own entries into the log database. As applications are not allowed to write directly to the log file, custom events are generated using an `ioctl` interface to the kernel driver. One use of user-entries includes marking events related to software installation, making it easier

to determine what files and registry keys are created/modified during installation. This feature also provides a general purpose logging facility.

The API for user-entries is `winresmon_userlog(logdata, length)`. It signals the trace generator to record the *logdata* to the log database in binary form. The ownership information of a user log entry (i.e., the program pathname and process) is always recorded.

Figure 2 shows a simple wrapper for an installer program which generates custom installation events. In this example, the *logdata* consists of the name of the software and path of the installer program. Invoking the wrapper as "`C:\ins-wrapper.exe photoshop_cs2 H:\setup.exe`", generates *logdata* containing "`INBGH:\setup.exe|photoshop_cs2`" and "`INEDH:\setup.exe|photoshop_cs2`".

```
#define MAGIC_INSTALL_BEGIN "INBG"
#define MAGIC_INSTALL_END "INED"

int main (int argc, char **argv)
{
    char buff[256];
    if (argc != 3) {
        printf("example: %s software_name\n",
            " c:\...\installer.exe",
            argv[0]);
        exit(1);
    }
    buff[sizeof(buff)-1] = '\0';
    _snprintf(buff, sizeof(buff)-1,
        MAGIC_INSTALL_BEGIN "%s|%s",
        argv[2], argv[1]);
    winresmon_userlog(buff, strlen(buff));
    system(argv[2]); // fork and wait
    _snprintf(buff, sizeof(buff)-1,
        MAGIC_INSTALL_END "%s|%s",
        argv[2], argv[1]);
    winresmon_userlog(buff, strlen(buff));
    return 0;
}
```

Figure 2: A sample installer wrapper.

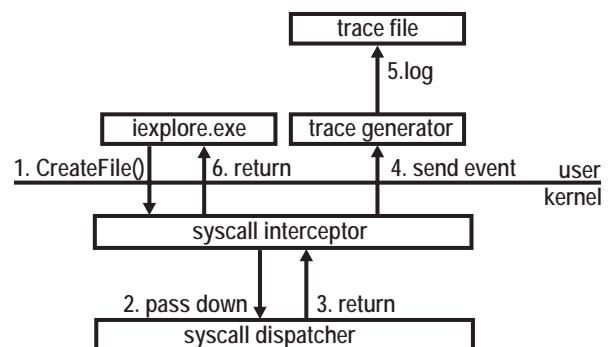


Figure 3: Overview of how the logger works.

Implementation

Figure 3 gives an overview of how the logger works. The information flow is as follows:

³Although an analyzer can include constraints on log's timestamp as conditions in the *selection condition*, an explicit time interval specification is less complex.

1. A sample program `ieplorer.exe` calls `CreateFile("C:\WINDOWS\system32\Macromed\Flash\Flash8.ocx", READ, ...)`.
2. This is intercepted by our system call interceptor which passes the parameters to the original syscall handling routine.
3. The syscall handling routine returns the file handle.
4. The handle, together with the system call parameters are then sent to the trace generator.
5. As the call is successful, the trace generator updates the file handle/path lookup table. Since "foo" is a relative path, the trace generator concatenates "foo" with the current directory and add it to the trace.
6. The handle is returned to `ieplorer.exe` and operation continues as per normal.⁴

System Call Interception

The overview of the logger in Figure 3 shows that resource usage is captured by intercepting system calls. In practice, this is actually more complex. Rather than defining and using the actual system calls, the Microsoft Windows API is described at a level higher than the operating system using the Win32 API. Although most programs use Win32, they can use the native API [4] directly. The native system call interface API is unfortunately not well documented and supported. Furthermore, the view of the operating system at the native API level is not quite the same as at the Win32 level. This means that intercepting system calls at what regular programs might think of as the Microsoft Windows API is problematic, and there could be discrepancies and mismatches between API use at the Win32 level and native level. To ensure accuracy, WinResMon intercepts system calls at the native API level.

WinResMon implements the syscall interceptor by means of a kernel mode driver. The driver captures syscall requests made by a process by "hooking" the native system calls as described in [5]. Appendix 1 lists the system calls intercepted by WinResMon. We found these to be the most common system calls arising from file, registry, IPC, synchronization, and process operations.⁵ To gather information about processes, WinResMon uses a simpler method. Microsoft Windows NT exports a set of process callback functions in the kernel space [6]. WinResMon makes use of `PsSetCreateProcessNotifyRoutine()` and `PsSetLoadImageNotifyRoutine()` which notifies the call back function during process creation or termination.

Event Handling

When sending the data from the kernel to the user space, it is inefficient to send every event as it

⁴Step 6 and 4 are independent. Thus, depending on the scheduler, step 6 is not necessarily executed after step 4.

⁵Since Microsoft Windows is closed source and the native API is only partially documented, it is difficult to make any guarantees about completeness.

arrives. WinResMon's implementation therefore makes use of double buffering. When the size of a buffer reaches a threshold, WinResMon sends the contents of the buffer to the user space and switches to another buffer to continue logging in the kernel space. This strategy tries to ensure that most, if not all, events are captured and reduces the system overhead due to context switching. Again, due to the undocumented nature of the kernel, we can not claim to capture all events.

In the prototype implementation, the kernel and user space communicate through an ioctl mechanism. Ioctls are used to define a protocol to synchronize data transfer between the driver and user level trace generator. WinResMon can also be extended into a remote-monitoring tool.

```

struct trace_struct *handle;
struct trace_entry *entry;

trace_handle = trace_select(
    "type==\"registry\" && "
    "prog_path=~\"/ieplorer.exe$/\"",
    "time, registry.path",
    "-ld TO NOW",
    FORWARD);
if (trace_handle == NULL)
    exit(1);
while ((entry = trace_next(trace_handle))
    != NULL) {
    printf("time=%s, registry=%s",
        entry->fields[0],
        entry->fields[1]);
}
trace_close(trace_handle);

```

Figure 4: A sample analyzer.

Writing Custom Analyzers

This section demonstrates how to write custom analyzers on top of the WinResMon framework by means of examples.

- An analyzer to show all the programs which read `C:\foo.txt` after 2005/1/1 could use the following query:

```

trace_select (
    "file.path == \"C:\\foo.txt\"",
    "prog_path",
    "2005-1-1 00:00:00 TO NOW",
    FORWARD);

```

- A more complicated example asks, "What's the most recent execution of `msnmsgr.exe` before this boot?" First determine the last *shutdown_time* with the query:

```

trace_select (
    "sysevent.type == \"shutdown\"",
    "time",
    "OLDEST TO NOW",
    BACKWARD);

```

The next query gets the whole process creation event for `msnmsgr.exe`:

```

trace_select ("proc.childname =~ "
  "\"/^.*\\msnmagr.exe$/\"",
  NULL,
  "OLDEST TO last_shutdown_time",
  BACKWARD);

```

- Figure 4 shows a code fragment from a simple analyzer which searches for all the registry keys opened by Internet Explorer.

WinResMon issues the query with selection type == "registry" && prog_path =~ "/iexplorer.exe/" and projection on time and registry.path. After correctly obtaining a trace_handle, it iterates over all selected records by using trace_next(). It prints all the fields, time and registry.path, for each selected trace entry. After iterating over all relevant records, it closes the handle.

Using WinResMon

The extended example below shows the use of WinResMon to solve software maintenance problems.

Yahoo Toolbar [7] adds tabbed browsing to Internet Explorer (IE) and adds various icons and links from within IE to different Yahoo services. The following walk-through illustrates monitoring the Yahoo toolbar throughout its entire life cycle. There are three stages: Installation, Program usage, and Uninstalling.

Installation

The provided installer refuses to run under a standard user account as it needs administrator privileges. Upon successful installation under an administrator account, it creates the following DLLs and keys:

DLLs

```

C:\ProgramFiles\Yahoo!\Companion\
Installs\cpn\yt.dll
C:\ProgramFiles\Yahoo!\Companion\
Installs\cpn\YTabBar.dll
...

```

Registry keys

```
HKEY\_LOCAL\_MACHINE\SOFTWARE\Yahoo
```

From our list of sensitive registry locations, we note the following.

Before Installation

Search Page:

```

http://www.microsoft.com/isapi/
redir.dll?prd=ie&ar=iesearch

```

Search Bar: -

After Installation

Search Page:

```

http://us.rd.yahoo.com/customize/
ycomp/defaults/sp/*http://www.yahoo.com

```

Search Bar:

```

http://us.rd.yahoo.com/customize/
ycomp/defaults/sb/*http://www.yahoo.com/
search/ie.html

```

Yahoo! Toolbar Helper:

```

02478D38-C3F9-4EFB-9B51-7695ECA05670 -
C:\ProgramFiles\Yahoo!\Companion\
Installs\cpn0\yt.dll

```

Among the changes made, Yahoo replaced MSN search as the default search engine.

Program Usage

Since the database is persistent, WinResMon logs all the events associated with Yahoo and preserves the information even if the system reboots. This helps analyze the behavior of Yahoo toolbar over a period of time.

Uninstalling

When uninstalling Yahoo toolbar, WinResMon observes that all the files have been removed. However, the registry settings it made are left unchanged. As a result, Yahoo remains the default search engine for the system.

WinResMon Overhead

To measure the performance overhead resulting from constant monitoring of systems with WinResMon, we first look at some worst case scenarios using micro-benchmarks consisting of only repeated system calls.

Our micro-benchmarks comprise of: seven benchmarks on file access, five on registry access, and two on process creation. All of these micro-benchmarks run on a Pentium 4 2.4GHz machine with 512MB running Microsoft Windows XP with SP2. The benchmarking procedure consists of first running the benchmarks on a clean Microsoft Windows XP (with SP2) to get the baseline performance. The next battery runs with WinResMon loaded. The last battery is run with FileMon [8] loaded for file access benchmarks and RegMon [9] loaded for registry access benchmarks. Each micro-benchmark repeats an operation n times. We performed each benchmark four times to get the average execution time. Tables 1, 2 and 3 show the average and standard deviation of the execution time in seconds.

The file access benchmarks consist of:

- (F₁) Open an existing file. The same filename is used every time.
- (F₂) Create a new file and delete it. A different filename is used every time.
- (F₃) Read 1 byte from a file. We ensure that the file is large enough so that EOF is never met for multiple reads.
- (F₄) Read 4,096 bytes from a file. The file size is a multiple of 4,096. When we reach EOF, we rewind to the beginning of the file.
- (F₅) Write 1 byte to a file. We start with an empty file.
- (F₆) Write 4,096 bytes to a file. When we reach EOF, we rewind to the beginning of the file.
- (F₇) Create a new directory and delete it. A different filename is used every time.

The benchmarks for registry access are:

- (R_1) Open an existing registry key. The same key is used every time.
- (R_2) Create a new registry key and delete it. A different name is used every time.
- (R_3) Create a new volatile registry key and delete it. `RegCreateKeyEx` is used with the `REG_OPTION_VOLATILE` option.
- (R_4) Query the value of a registry key. The type `REG_DWORD` is used.
- (R_5) Set the value of a registry key. The type `REG_DWORD` is used.

The benchmarks for process creation are:

- (P_1) Create a dummy console process and wait for its termination.
- (P_2) Create a dummy GUI process and wait for its termination.

Table 1 shows the execution time (in seconds) for the file access benchmarks. In order to avoid any extraneous overhead from the FileMon GUI, the window is always minimized during the experiments. It appears that FileMon does not capture all the operations during the performed micro-benchmark, though. For example, during the “Read 1 byte” test, 10 M events occurred, but FileMon only captured about 15K. During the “Create a new file” test, 600 K events occurred, but only about 18 K were actually captured. We observe that WinResMon, by contrast, captured all operations in all the tests.

Table 2 shows the execution time (in seconds) of the registry related benchmarks. This benchmark is conducted similarly to the file benchmark. It appears that RegMon also drops events. For example, during one of the “Query value” test, 1 M events occurred, but only 993,938 were actually captured by RegMon.

Note that FileMon, RegMon and WinResMon address different goals. FileMon and RegMon are meant for short term monitoring while WinResMon is

designed for long term use and is therefore always running in the background. These two benchmark comparisons merely give us a baseline on how WinResMon compares with other, similar monitoring software.

Table 3 shows the results of the process creation benchmark. The console process in the benchmark creates a dummy child process using the `CreateProcess()` function and waits for its termination using the `WaitForSingleObject()` function. The difference between a console program and a GUI program is that the GUI program uses the `WinMain()` entry function and is linked using the `/SUBSYSTEM:WINDOWS` option, while the console program uses `main()` and `/SUBSYSTEM:CONSOLE`. The measured overhead is quite small because process creation is a slower operation than file or registry access.

We would expect normal programs to have much smaller overhead than that of the micro-benchmarks because the micro-benchmarks are very system call intensive. Normal programs, such as our macro-benchmarks, typically make significantly fewer system calls, spending more time in the application rather than the kernel. Table 4 gives some macro-benchmark results which show the impact of WinResMon on the following normal programs: WinRAR, gcc, \LaTeX , and Lame. The benchmarks perform the following:

- WinRAR: compress a 150 MB file
- gcc: compile a 500 K-line C program
- latex: compile a 2,000-page \LaTeX file into PDF using the `pdflatex` program
- Lame: encode a 100 M wave file into a mp3 file.

The macro-benchmark results demonstrate that running WinResMon all the time is quite reasonable under typical usage.

Related Work

From a high level perspective, WinResMon differs from previous systems/tools in that it is:

File Operation	n	Clean	WinResMon	FileMon
(F_1) Open an existing file	1 M	20.457 ± 0.240	46.266 ± 3.271 (126.2%)	44.168 ± 0.279 (116.0%)
(F_2) Create a new file	100 K	53.004 ± 2.532	67.539 ± 0.469 (27.4%)	73.117 ± 0.265 (37.9%)
(F_3) Read 1 byte	10 M	14.277 ± 1.084	278.175 ± 14.282 (1848.4%)	107.414 ± 4.765 (652.4%)
(F_4) Read 4096 bytes	10 M	41.207 ± 0.133	328.203 ± 34.869 (696.5%)	138.816 ± 0.793 (236.9%)
(F_5) Write 1 byte	10 M	49.824 ± 1.160	388.050 ± 0.837 (678.8%)	172.422 ± 1.114 (246.1%)
(F_6) Write 4096 bytes	10 M	116.355 ± 0.716	448.933 ± 2.192 (285.8%)	212.828 ± 2.950 (82.9%)
(F_7) Create a new directory	100 K	46.546 ± 0.344	57.750 ± 9.565 (24.1%)	56.395 ± 0.282 (21.2%)

Table 1: Performance comparison on file access (in seconds).

Registry Operation	n	Clean	WinResMon	RegMon
(R_1) Open an existing key	1M	10.378 ± 0.039	35.324 ± 0.080 (240.4%)	361.438 ± 40.504 (3382.7%)
(R_2) Create a new key	100K	8.980 ± 0.037	13.769 ± 0.041 (53.3%)	134.879 ± 10.778 (1402.0%)
(R_3) Create a new temp key	100K	7.832 ± 0.045	12.750 ± 0.082 (62.8%)	142.961 ± 12.811 (1725.3%)
(R_4) Query value	1M	1.461 ± 0.009	27.203 ± 0.061 (1761.9%)	166.301 ± 4.406 (11382.7%)
(R_5) Set value	1M	22.890 ± 0.153	46.379 ± 0.090 (102.6%)	182.473 ± 7.272 (697.1%)

Table 2: Performance comparison on registry access (in seconds).

- *integrated*: since it monitors accesses on files and registry under one infrastructure
- *extensible*: system administrators can write their own custom modules to utilize the generated log
- *geared for log management*: system administrators can view resource access activities generated over time, and inspect their relationships with respect to software configuration and dependencies.

We briefly mention some other tools/systems below, and highlight the important differences with WinResMon.

FileMon [8] and RegMon [9] are file and registry monitoring tools, respectively. They monitor operations taking place on the registry or specified file system in real time. Although WinResMon shares the basic monitoring functionalities with these two tools, WinResMon's infrastructure is integrated, and its log database is designed to assist system administrators in inspecting software configuration and dependencies.

Strace [10] is a Linux/UNIX tool used to intercept and log system calls invoked by a process. There is also a Microsoft Windows NT port of strace [11] with similar functionality. WinResMon differs from strace in that it is focused more on resource usage (files, registry, etc.) rather than system calls. In Microsoft Windows, a system call viewpoint can be confusing since there are multiple levels of APIs which translate into the poorly documented native API.

Systrace [2] is a UNIX tool for sandboxing untrusted code. Unlike Systrace, which examines system-call sequences issued by the monitored processes and applies a specific security policy, WinResMon is meant as a monitoring tool to inspect resource usage and interactions among programs in a system.

DTrace [12], SystemTap [13] and LBox [14] are auditing and instrumentation systems on various UNIX operating systems. They are all event based auditing systems, performing a specific action only on a specific event. DTrace and SystemTap allow administrators to dynamically execute supplied code in the kernel when certain event occurs. LBox allows the kernel to notify a user space program when certain events happen. Both

LBox and WinResMon are designed for monitoring resource usage, while DTrace and SystemTap are designed for general system call instrumentation.

Conclusion

This paper presented the motivation, design, implementation and usage of WinResMon. Its main use is to inspect resource access and software dependency issues in Microsoft Windows environments. As WinResMon is extensible, system administrators can also build tools using WinResMon for custom queries and system analysis. Benchmarking shows that WinResMon is reliable and is comparable to other popular tools.

Future work is to increase the usability and robustness. We would also like to ensure that logging is as comprehensive as possible taking into account the undocumented and unsupported nature of the APIs in the Microsoft Windows NT kernel. We would also like to further increase the efficiency of the logging mechanism.

Acknowledgments

We acknowledge the support of the "Defence Science and Technology Agency" and "Temasek Laboratories." We also would like to thank Amy Rich for many useful comments and suggestions in improving the paper.

Author Biographies

Rajiv Ramnath is currently a final year student in the School of Computing at National University of Singapore. His interests include operating systems and computer security. He can be reached at rajivram@comp.nus.edu.sg.

Sufatrio holds a B.Sc. from University of Indonesia and an M.Sc. from National University of Singapore. He is currently a Ph.D. student in the School of Computing and an associate scientist in Temasek Laboratories at National University of Singapore. His interests include intrusion detection systems and infrastructure for secure program execution. He can be reached electronically at tsufat@nus.edu.sg.

Roland Yap obtained his Ph.D. from Monash University. He is currently an associate professor in the School of Computing at National University of

Process Operation	n	Clean	WinResMon
(P_1) Create a console process	10K	35.488 ± 0.071	37.855 ± 0.150 (6.7%)
(P_2) Create a GUI process	10K	34.641 ± 0.044	36.938 ± 0.097 (6.7%)

Table 3: Performance of process creation (in seconds).

Test Case	Clean	WinResMon
WinRAR	224.443 ± 0.542	226.524 ± 3.502 (0.9%)
gcc	26.265 ± 1.219	26.973 ± 0.968 (2.70%)
TEX	27.211 ± 0.473	27.498 ± 0.981 (1.1%)
Lame	45.631 ± 0.538	45.662 ± 0.534 (0.6%)

Table 4: Performance of macro-benchmarks (in seconds).

Singapore. His interests include systems security, operating systems, programming languages, and distributed systems. He can be reached electronically at ryap@comp.nus.edu.sg.

Wu Yongzheng holds a B.Comp. from National University of Singapore. He is currently a Ph.D. student in the School of Computing at National University of Singapore. His interests include systems security and operating system. He can be reached at wuyongzh@comp.nus.edu.sg.

Bibliography

- [1] <http://www.rpm.org/>.
- [2] Provos, N., "Improving Host Security with System Call Policies," *USENIX Security Symposium*, pp. 257-272, 2003.
- [3] Wang, Y. M., R. Roussev, C. Verbowski, A. Johnson, M. W. Wu, Y. Huang and S. Y. Kuo, "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management," *Large Installation System Administration Conference*, pp. 33-46, 2004.
- [4] Nebbett, G., *Windows NT/2000 Native API Reference*, Macmillan Technical Publishing, Indianapolis, 2000.
- [5] <http://www.ddj.com/184410109>.
- [6] Microsoft MSDN, "Process Callbacks," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/Kernel_r/hh/Kernel_r/k108_a0f7bff2-270e-41fb-87d4-d8d533aa0bef.xml.asp.
- [7] <http://toolbar.yahoo.com/>.
- [8] <http://www.sysinternals.com/Utilities/Filemon.html>.
- [9] <http://www.sysinternals.com/Utilities/Regmon.html>.
- [10] <http://www.liacs.nl/wichert/strace/>.
- [11] http://www.bindview.com/Services/RAZOR/Utilities/Windows/strace_readme.cfm.
- [12] Cantrill, B. M., M. W. Shapiro and A. H. Leventhal, "Dynamic Instrumentation of Production Systems," *USENIX Annual Technical Conference*, pp. 15-28, 2004.
- [13] Prasad, V., W. Cohen, F. Eigler, M. Hunt, J. Keniston, B. Chen, "Locating System Problems Using Dynamic Instrumentation," *Linux Symposium*, Vol. 2, pp. 57-72, 2005.
- [14] Wu, Y. Z. and R. H. C. Yap, "A User-level Framework for Auditing and Monitoring," *Annual Computer Security Applications Conference*, pp. 95-105, 2005.

Appendix 1: List of Intercepted System Calls

Table 5 lists the Microsoft Windows NT native-level system calls intercepted in our current implementation.

File	
ZwCreateFile	opens or creates a new file
ZwOpenFile	opens an existing file
ZwDeleteFile	deletes a file
ZwReadFile	reads from an open file
ZwWriteFile	writes to an open file
ZwQuerySystemInformation	queries for information internal to the system
Registry	
ZwCreateKey	opens an existing key or creates it if it does not exist
ZwDeleteKey	deletes a key
ZwOpenKey	opens an existing key
ZwQueryKey	provides information about the size and number of subkeys (if any)
ZwQueryValueKey	provides the value of a registry key entry
ZwSetValueKey	creates or replaces a registry key's value entry
ZwDeleteValueKey	deletes a registry key's value entry
Process	
ZwTerminateProcess	terminates a process and all its threads.
Synchronization Object (Mutex)	
ZwCreateMutant	creates a mutex or opens an existing mutex
ZwOpenMutant	opens an existing mutex
Synchronization Object (Semaphore)	
ZwCreateSemaphore	creates a semaphore or opens an existing semaphore
ZwOpenSemaphore	opens an existing semaphore
Synchronization Object (Event)	
ZwCreateEvent	creates a new event or opens an existing event
ZwOpenEvent	opens an existing event
Synchronization Object (Waitable Timer)	
ZwCreateTimer	creates a new timer object or opens an existing timer object
ZwOpenTimer	opens an existing timer object
IPC (Named pipe)	
ZwCreateNamedPipeFile	creates a named pipe
IPC (Mailslot)	
ZwCreateMailslotFile	creates a mailslot

Table 5: Intercepted system calls.

Appendix 2: Example of Log Priorities for Trace Compaction

The following example shows a fragment from a log configuration which assigns priorities on FILE resource-type entries. We map log entries into priority values ranging from 1 (least important to retain) to 5 (most important to retain).

```
# File Section
# Format:
# Type      Action  Object                                Mode      Priority
FILE      Read    *                                    *          1
FILE      Write   *                                    *          1
File      Open    C:\Windows\Temp\*                  *          1
File      Open    C:\Windows\System32\*              RW|WO      5
File      Open    C:\Windows\System32\*              RO          4
File      Open    C:\Windows\*                       RW|WO      4
File      Open    C:\Windows\*                       RO          3
File      Open    C:\ProgramFiles\*                  RW|WO      3
File      Open    C:\ProgramFiles\*                  RO          2
File      Open    C:\Documentsand Settings\LocalSettings\
    {Temp\*|Temporary Internet Files\*}    *          1
File      Open    *                                    RW|WO      2
File      Open    *                                    RO          1
File      Delete  C:\Windows\Temp\*                  *          1
File      Delete  C:\Windows\System32\*              *          5
File      Delete  C:\Windows\*                       *          4
File      Delete  C:\ProgramFiles\*                  *          3
File      Delete  C:\Documentsand Settings\LocalSettings\
    {Temp\*|Temporary Internet Files\*}    *          1
File      Delete  *                                    *          2
....
```

LiveOps: Systems Management as a Service

Chad Verbowski – Microsoft Research
Juhan Lee and Xiaogang Liu – Microsoft MSN
Roussi Roussev – Florida Institute of Technology
Yi-Min Wang – Microsoft Research

ABSTRACT

Existing Management Systems do not detect the most time-consuming and technically difficult anomalies administrators encounter. Oppenheimer [25] found that 33% of outages were caused by human error and that 76% of the time taken to resolve an outage was taken by humans determining what change was needed. Defining anomaly detection rules is challenging and often cannot be shared across organizations. It requires a deep combined knowledge of the software, workload, system configuration, and tuning parameters specific to the workload and overall distributed application topology.

We present LiveOps, a scalable systems and security management service based on auditing the interactions between applications and the persistent state they use [33]. This approach simplifies identifying security vulnerabilities, performs compliance auditing, enables forensic investigations, detects patching problems, optimizes troubleshooting, and detects malware/intrusions. The service enables knowledge sharing across organizations and administrative boundaries and allows for seamless integration between analysis results from disparate management products that build on it. Our configuration-free agent collects all read and write access to registry entries, files, binaries, and process creation. The agents streaming lossless compression creates log files of only 20 MB per day containing an average of 45 million events. The scalable LiveOps back-end service can analyze 1000 machine days of logs in 30 minutes. LiveOps agents have been deployed on 1149 machines from home systems to corporate desktops, including 381 production MSN servers across 11 sites.

Introduction

MSN System administrators spend a third of their time managing system configuration despite the use of state-of-the-art systems management solutions. This not only reduces the number of systems that a single administrator can effectively manage, but is a significant liability in overall system reliability.

Several examples of configuration errors impacting system reliability are described in [33], where at one MSN site, 70% of persistent errors were found to be persistent state (PS) related, and 28% of support calls at a large software company's help desk were configuration related. Furthermore, a lack of understanding about the impact configuration changes have on critical applications can delay the deployment of critical security patches [2, 28, 34], and has been found to be responsible for 76% of the time taken to resolve datacenter outages [25].

Existing systems management solutions are difficult to implement, expensive to deploy, and are largely ineffective at addressing the most costly systems and security management problems. Furthermore, for scalability and performance reasons they do not track all changes, or the interactions between applications, users and PS. Administrators must manually apply their

experience and application knowledge to filter the subset of PS changes recorded by these systems to identify only the PS changes impacting their applications.

Defining anomaly detection rules based on application generated events [31, 40] is challenging and often cannot be shared across organizations. It requires a deep combined knowledge of the software, workload, system configuration, and tuning parameters that are specific to the workload and overall distributed application topology.

Software developers and systems management solution vendors are restricted to creating rules that are based on the available application events, and that are locally tunable or independent of workload, system configuration, and distributed application topology. Administrators are left with the challenge of determining the appropriate tuning parameters for existing rules and are required to identify and codify the bulk of rules needed to manage their environments. Sharing rules between organizations is further complicated by variations in tuning parameters and rule definition languages.

We present LiveOps, a scalable systems and security management service based on auditing the interactions between applications and persistent state they use

[33]. This approach simplifies identifying security vulnerabilities, performs compliance auditing, enables forensic investigations, detects patching problems, optimizes troubleshooting, and detects malware/intrusions.

The LiveOps architecture consists of agents running on the managed systems that report logs to a back-end service which processes the data and provides extensible interface for generating of web reports, alerts, or integrating with other management products. Our configuration-free agent collects all read and write access to registry entries, files, binary loads, and process creation. The agent's streaming lossless compression creates log files of only 20 MB per day containing an average of 45 million events. The scalable LiveOps back-end service can analyze 1000 machine days of logs in 30 minutes

The key contribution of LiveOps is scalable and complete configuration monitoring, comprehensive anomaly detection through cross machine and cross time baseline analysis in an organization, and globally across all machines reporting to the service, enabling self-tuning of rules with respect to workload and topology. Secondly LiveOps' ability to manage 4000+ monitored machines via a single back-end server enables it to be run as a service supporting multiple organizations.

As a service, LiveOps provides immediate benefit to all subscribers when new rules are developed. Using a service rather than a locally installed management system eliminates the cost of maintaining a systems management solution, enables IT departments to draw on the expertise of other administrators, and potentially alerts the original developers of a problematic application. When rare and difficult issues are encountered, the service provides the specific management details that enable collaboration with experts that can help resolve the issue. Furthermore, after an issue is resolved the relevant centralized data can be annotated to benefit other users experiencing a similar problem.

We present our experiences and analysis of the alerts and reports collected from running LiveOps in MSN. LiveOps agents have been deployed on 1149 machines, from home systems to corporate desktops, and including 381 production MSN servers across 11 sites. In the next section, we describe the motivation for creating LiveOps and the related work comparing it with existing management approaches. We subsequently describe the LiveOps architecture and present the management scenarios that LiveOps addresses, provides sample reports, and a summary of results obtained from analyzing the reports from MSN production servers. We then present a data analysis of the PS used in a production environment, and an analysis of the feasibility of classifying them. Finally, we conclude.

Motivation and Related Work

The creation of LiveOps was motivated by the discovery that infrequently occurring unique configuration

issues have a large reliability impact, require the most administrator time, and are the most technically challenging to resolve. Traditional management products [21] are largely ineffective at managing these problems because they rely on the monitored applications to log events [31, 40] when they are malfunctioning.

It is unrealistic to expect developers of the application to have correctly anticipated all possible problems that may occur with the application internally – from integrating with other applications, OS interactions, and interacting with distributed systems such as databases, firewalls, web servers, and network related services. Events logged by applications are best used for automating responses to frequently occurring problems that are well understood, but seldom provide sufficient insight for administrators to enable quick resolution of issues unanticipated by the original application developers. The following example of a problem at a large MSN site illustrates how understanding the impact of individual PS changes on an application enables quick problem resolution:

An administrator was assigned to resolve an intermittent web page issue with a large online site. During the course of solving the problem, a critical configuration file was inadvertently deleted. After the intermittency issue was resolved, the site appeared fixed and the issue was closed.

After 18 hours it was discovered that the site was experiencing a partial outage. Investigation of the outage involved teams of engineers, and lasted for 27 more hours, before the originally modified configuration file was discovered as the root cause. With a record of what PS had been changed on each machine, the investigators could have quickly and easily identified the configuration file as a root cause candidate.

The next example illustrates how failing to verify the complete set of files and settings modified after a patch installation can prevent diagnosis of configuration problems:

A patch was deployed on servers across a large MSN site. Later it was found that the site was experiencing a partial outage after the installation, where some users experienced poor performance or received an error message after refreshing their web page two out of five times.

Since the site load balances incoming requests over a large pool of servers, it was difficult to determine which servers were causing the problem. It took two engineering teams approximately 72 hours to determine that the root cause was that one of the servers had only received partial settings during the recent patch install.

The LiveOps approach is to manage configuration from the OS perspective of interactions between running processes and PS. This differs from asserting correctness constraints on subsets configuration for

the system as a whole, as done with CFEngine [5], because using interactions enables consideration of the subset of PS that are used by each application. Overall this reduces the volume of PS to consider because less than 15% of non-temporary registry and files are typical in daily use [32].

We believe this is more effective than the traditional approach of analyzing application logs for the following reasons:

1. It is a non-participatory model, meaning that all changes made by applications are tracked regardless of the APIs they use or logs they create.
2. Only a few well-defined event types from a single source are required for monitoring all applications, as opposed to multiple event logs containing application specific events.
3. If the specific root cause of a problem is not obvious the information provided reduces potential root cause candidates, making problems easier to solve.
4. When the root cause of the problem is found, future incarnations of the problem can be avoided through knowing which process was used by whom at which specific time to make the breaking change.

Another approach to configuration management is to eliminate the need to identify the root cause of problems by following a non-traditional approach of designing applications with the expectation that they will fail [4], as opposed to attempting to eliminate all possible errors. This is achieved by making them quick to install and restart, and by creating software probes that remotely monitor the application to detect failures and restart the application when failures are detected. While this approach minimizes the time spent by operators investigating problems, the lack of root cause understanding means that future occurrences of the problem cannot be avoided.

The implicit assumptions are that problems will not simultaneously affect large numbers of machines, and that problems affecting a machine will be relatively infrequent. These may not be valid assumptions if DDOS attacks are made on the infrastructure. Furthermore, this approach requires that all failures be detectable by the probes.

LiveOps provides a critical missing component of the software configuration management cycle (shown in Figure 1) by detecting all system changes, verifying approved requests, and alerting upon unknown modifications. Closing this loop is becoming increasingly important in order to identify unwanted changes, malware for example, and to fulfill auditing requirements [27, 29].

Managing servers and desktops through a management service has several advantages over individual companies or departments maintaining a local management infrastructure. These are:

1. Current and historic information about the managed systems is available and accessible despite local outages and system failures, and that it provides a quick and easy method of sharing detailed information with support professionals and other problem domain experts without local network or system access.
2. The service enables correlation of system activities across large numbers of managed hosts to identify known good and known bad values.
3. The service enables application and OS developers to receive details on how their software is being configured and used which can help identify problems, and drive future product enhancements.
4. The service provides a centralized, globally available, and integrated system for application developers, administrators, and support professionals to add knowledge on specific PS, regarding optimal values, and new analysis rules to identify and resolve problems so future instances will not require detailed investigation.
5. The service becomes a management platform for new reports so problem analysis techniques can be added to the service's back-end servers without requiring software or configuration updates to the agents running on managed systems.

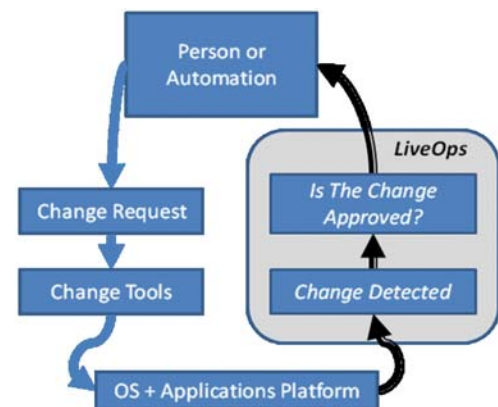


Figure 1: Typical Change Management Process showing how LiveOps closes the loop.

Architecture

In this section, we present an overview of the LiveOps system architecture. Figure 2 describes the system architecture and data flow of:

1. Our low-level server or desktop agent that logs all registry, file, binary load, and process creation interactions, compresses the trace events into log files, and uploads them for analysis [33]
2. The LiveOps back-end service that processes and archives the uploaded log files
3. An extensible web interface that supports retrieving reports, programmatic data access, and integrating with existing notification mechanisms.

Our implementation does not require any changes to the core operating system or any applications specific changes or configuration

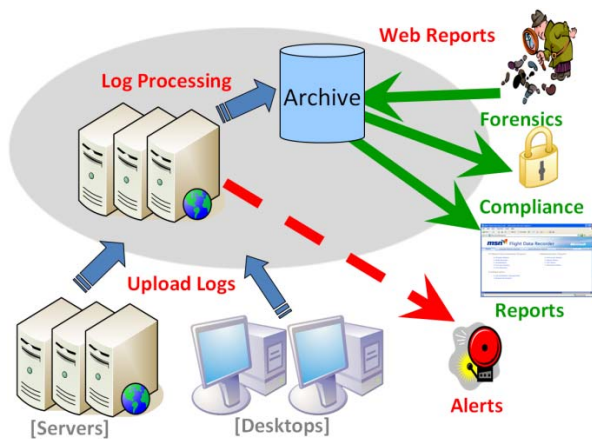


Figure 2: LiveOps System Architecture and data flow.

Figure 3 describes the processing flow of the back-end LiveOps servers, detailing the analysis performed by the daemons on newly received logs. Anomaly detection rules, such as checking for policy violations or correlating an observed change with a planned change work order, are written in standard C/C++/C# code and

compiled into a dynamically loadable Query Module (QM). At startup the daemons load the QMs, passing each QM a reference to the newly uploaded logs.

Internally, a QM may integrate with external IT data sources, such as change management databases (CMDB) or change policy definitions. The QM then detects anomalies such as identifying patterns within the logs, correlating log activities with CMDB work items, or using change policy definitions to identify policy violations. All QMs write alerts to a common API that stores them in the Alert Database. An Alert Notification daemon reads the alerts and sends notifications through Instant Messenger, email, and RSS feeds. A QM may also maintain an internal database, such as the baseline database used to store a history of application interaction with PS. The baseline QM uses its database to detect deviations and log them as alerts. Once log files have been processed by all QMs they are moved into a central archive. Administrators can create ad-hoc queries against all archived data using the web service that exposes the LiveOps query API. This web service is used by the LiveOps web server to generate HTML reports, and can also be used by other products to integrate with LiveOps data.

In addition to providing access to the information, LiveOps also provides contextual information about each alert and the settings it describes. Three stages of annotations:

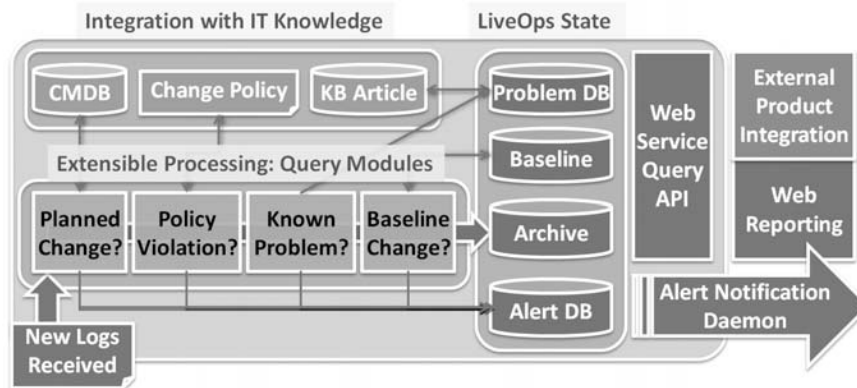


Figure 3: LiveOps analysis framework.

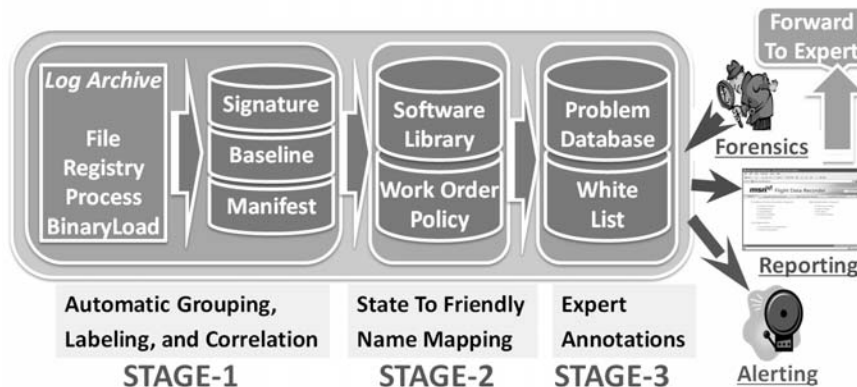


Figure 4: LiveOps annotation of information.

1. *Automatic*
2. *Pre-defined context mapping*
3. *Expert*, as described in Figure 4, are applied to the alerts and individual file and settings as they are read from the web interface

For example, let's consider the annotations added for an alert that describes a new application installation on a monitored machine. The first level provides statistical information such as how many other machines have this application installed, and the most common versions of files and values of settings associated with it. The second level of annotation compares the binary hash of the installed files with a database that indexes company name, application name, and version information based on binary hashes. The final level of annotation adds any comments or rating information provided by other users of the system.

If this installation was known to be spyware or malware, the final annotations could include support articles that describe how to deal with the installation. The quality of this information grows as more machines report their data to the service, and as more users provide comments and ratings on the alerts and individual file and settings entries. It also provides valuable insight into the application vendors about how consumers configure and use their products, along with insight into the real-world problems their products have.

LiveOps Scenarios

LiveOps was created within the management philosophy that administrators understand the PS of a running system, all changes made to the PS, and the scope of PS interactions for correctly running processes. This is required in all administrative management scenarios which are described in the following subsections. Each section contains a description of the LiveOps reports, how they are generated, and a summary of results from running LiveOps on production servers at MSN.

The section on "Understanding Changes" describes how LiveOps improves the change management processes used by most IT organizations by showing how it can be used to:

1. Verify that changes are correctly made and to discover unauthorized changes
2. Identify unapproved processes
3. Discover the impact and frequency of changes impacting critical systems and applications
4. Identify sensitive information that is being copied to removable devices and network locations
5. Analyze and account for the PS that are often left behind by software installation and removal programs

The section on "Managing Abnormal System Activity" shows how LiveOps can significantly reduce troubleshooting time by:

1. Detecting known problems

2. Providing contextual information about what PS processes and users were accessing
3. Identifying processes that have become stale from changes to PS

"Enforcing Best Practices" describes how LiveOps can be used to improve the best practices followed by administrators by identifying:

1. When users are logging in to systems, potentially unnecessarily
2. Daemons running under local user or domain user accounts instead of the system account

Scenario: Understanding Changes

The typical change management process followed by many IT organizations is:

1. Identify change to be made and create a work order
2. Verify correctness of change in test environment with test workloads
3. Deploy the change to subset of servers and monitor for problems caused by real-world workloads
4. Broadly deploy the change and close the work order

The three main challenges with implementing this process are:

1. Defining the specific changes expected
2. Auditing the changes as they are made to the systems
3. Identifying the impact of the change to understand the applications that must be tested and verified

Without LiveOps 'Change Management' reports defined in this section, scheduled changes are often not understood and therefore their impact is not well known. For example, work orders may define a change as applying a SQL Server Service Pack 1 update, rather than listing the specific files and settings being modified. Consequently, understanding if the change has been correctly and completely applied to all required servers is often based on unreliable and incomplete methodologies, such as:

1. Sampling each affected server for one of the known files in the change
2. Relying on the administrator notes about what changes were made and when
3. By examining logs that may not fully describe the change

Often times the software installation for patches and applications does not maintain an accurate record of the PS being created or changed on the system, and consequently their uninstall programs can leave PS behind, or even corrupt the remaining applications on the system. In a later section, we present a case study of software installation and removal that demonstrates the significance of this problem.

Furthermore, current auditing techniques can only detect changes where existing changes are recorded by the affected applications and system logs,

or those which are manually reported by administrators. For example, if an administrator makes multiple configuration changes in an attempt to troubleshoot a problem and forgets to roll back one of them, this change will not be recorded and may cause a future problem on the server.

In addition to auditing changes as part of the change management process, there is a growing need to audit all interactions on systems. Several regulatory bodies require maintaining strict audit logs of changes made to systems [27, 29]. Also, new consumer protection laws enforced in Japan and being drafted in other countries, now make businesses liable for Personally Identifiable Information (PII) that is leaked or stolen from their companies [13, 14]. In addition, the growing trend to outsource business functions that relate to Intellectual Property (IP), such as the source code used to develop software, provides a need for auditing IP and PII for when it is accessed, modified, or copied.

To improve the change management process and facilitate regulatory auditing, LiveOps provides reports that describe critical changes; document unauthorized applications and change impact analysis; and list sensitive data copied to network and removable devices. The remainder of this section describes these reports in more detail and summarizes our results from generating them in MSN datacenters.

Report: Critical Changes

LiveOps defines critical changes as:

1. Unexpected program execution
2. Modifications made to PS, used by the operating system and line of business (LOB) applications

The remainder of this section defines the alerts used to generate critical change reports, and describes the distribution of alerts detected for a one-month sample of LiveOps data.

Datacenter reviewers of the report can manually associate the processes needing approval with work order systems. LiveOps can be used to verify configuration changes, potentially in an automated way, by

correlating planned changes with the process name, user name, and time of the changes made. LiveOps marks alerts for processes requiring approval that run during ‘Lockdown’ periods by integrating with the lockdown schedule maintained externally to LiveOps by operations managers.

Changes to PS used by the OS and LOB applications must be authorized and controlled to avoid reliability and availability problems. Changes made to management applications on these servers are also important because they control the insight administrators have into the performance, reliability, and change management behavior of the server. Incorrect changes to management applications may reduce or eliminate visibility, and may expose the system to attackers.

While it is important to understand the critical changes affecting systems, not all writes to PS are interesting to administrators. To identify the critical changes from all writes to PS the following nine prioritized classifications are used to label each entry:

- **Problem:** Indicates a known problem – results from the existence or removal of this PS.
- **Install:** PS added as part of an installation or upgrade.
- **Setting:** Changes made to configuration PS.
- **Content:** Web pages, images, and user data.
- **Management Change:** Installation, patching, or configuration changes made to the management applications running on the system.
- **Unauthorized:** Installation of prohibited applications, or configuration changes to prohibited values.
- **User Activity:** PS modified as a result of users logging in or running window applications.
- **Noise:** Temporary or cached PS.
- **Unknown:** Unclassified PS.

The classification process involves associating each match of a substring contained in a classification rule to the PS name contained in each change event. Matches to classification substrings with higher priority take precedence over those with lower priority. For

State Classification	State Grouping		Process Grouping of State Changes			
	All	Distinct	Daily Instances	Daily Distinct	Monthly Distinct	Average Per Machine Daily Distinct
Problem	0	0	0	0	0	0
Install	104,149	16,947	810	155	69	4
Configuration	176,300	3,340	399	86	22	3
Content	16,261,721	1,593,100	9,513	50	1	3
Management Change	57,145	864	234	79	11	2
Unauthorized	4,206	634	14	9	3	2
User Activity	1,221	189	96	33	4	3
Noise	104,109	1,727	909	66	14	2
Unknown	39,715	2,613	534	60	13	3
TOTAL	16,748,566	1,619,414	12,509	538	137	22

Table 1: Critical changes for one month of production server logs across 34 machines showing individual changes and changes grouped by process.

example, a PS name matching both ‘User Activity’ and ‘Install’ classification substrings will be labeled as ‘Install.’ From examining 28 days of change activity from 34 systems, we identified 1 to 20 substring rules for each classification. Administrators can update the classification rules as needed, however, Table 1 shows that the initial rules cover all except 0.2% (39 k/16.7 M) of the PS observed.

Table 1 shows a summary of the changes observed across one month of traces from 34 production servers. It shows that 16.7 M individual changes were made to 1.6 M distinct PS entries, meaning that on average each PS was changed 10 times. Considering an overall average of 300 k PS entries per machine, this means that only 16% (1.6 M distinct state grouped changes / (300 k PS * 34 machines)) of the PS was modified during this period.

Although 1.6 M changes are too many for an administrator to review we can significantly reduce the items to review if we assume that each process instance is updating a set of related PS. For example, an installation program will add thousands of Registry entries and hundreds of files during the installation of a single application.

Table 1 shows that there is an $O(10^2)$ reduction in changes if daily change process instances are considered instead of distinct state. This can be further reduced to $O(10^3)$ by considering only changes made by daily distinct processes, however, comparing monthly distinct with daily distinct we reduce the number of changes to consider by only half. For example, in MSN it is common for the same patch installer, KB-123.exe, to be run on all 34 systems, which can be presented as one distinct kb-123.exe entry, providing the administrator with the ability to ‘drill in’ to see the affected machines and individual PS changes. If we consider the average daily distinct changes for each machine, rather than the overall distinct changes, we find that there are only $O(10^1)$.

The LiveOps lockdown reports show configuration changes such as installations, patching, changes to LOB, OS, or management applications during periods when changes are prohibited on systems. Lockdown reports from five properties were analyzed for nine lockdown ranges over a seven month period. Figure 5 shows that all properties had lockdown violations during at least one period; two properties had violations in eight of nine lockdown periods. Violations ranged from minor issues such as running diagnostics on systems, to more severe changes like installing service packs and new applications. We expect identifying and attributing lockdown violations to specific administrators will be a strong deterrent.

Report: Unauthorized Applications

Only approved processes should be running on datacenter systems. Similar to systems such as TripWire [16], LiveOps uses a predetermined list of approved

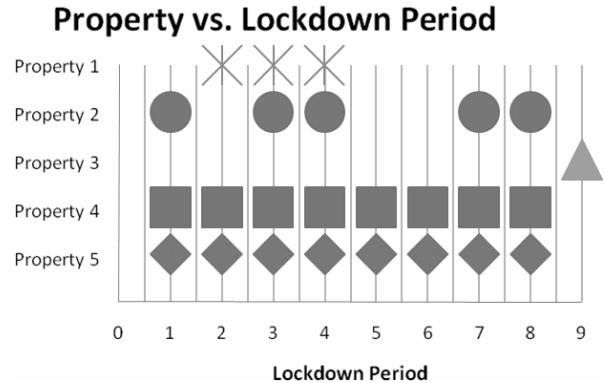


Figure 5: A marker for each property with at least one violation during a Lockdown period.

and unapproved applications to identify unauthorized processes on servers. The list contains the following details for each of the approximately 200 distinct process names observed on production servers, and 1300 distinct process names observed across all systems including desktop and home machines:

- **Approved** – Marks the process as approved (or not) for running on production systems, or requires a work order before running it.
- **Type** – Indicates if this process is a command line tool, daemon, or interactive window application. Reports will use this classification to determine if daemons are not running under local system credentials, if window applications are being remotely launched, or if local logins are made when administrative activities could be done remotely.
- **Category** – Indicates the intended use of the process
 - **LOB:** Created specifically for a business need, such as the infrastructure for a web site.
 - **OS:** Processes that run as part of the OS as required components, such as lsass.exe.
 - **Desktop:** A corporate desktop application like an email client. Applications that are used in server environments but designed for home or desktop use may not meet the stringent security requirements of a server environment.
 - **Dev:** A developer tool such as a compiler.
 - **Home:** For home entertainment, such as a game.
 - **Mgmt:** Management products and tools used to manage systems like agents or event log tools that ship with the OS. Instances of diagnostic tool processes can indicate server problems may exist, and instances of configuration tools indicate that prior authorization should be granted.
- **Function** – Describes what this process can potentially do on the system. This classification is used by reports that highlight users running

processes that could make changes to the system's PS.

- **Setup**: Can be used to install or remove applications.
- **Patch**: Updates existing applications.
- **Config**: Can be used to modify the configuration of the system.
- **Diag**: Used for retrieving diagnostic information from the system. Examples are ping and traceroute.
- **Viewer**: Displays read-only data, from files or other sources. Tools like 'winver.exe' fall into this category.
- **WriteData**: Indicates the process can create data such as log files like the LiveOps agent, or can modify existing data like notepad.exe.
- **"Fun"**: have no business value, and used only for entertainment. Examples are games and DVD playing applications.
- **System**: Required for operation of the system. Some examples are lsass.exe and csrss.exe.
- **Runtime**: The process hosts other applications. Examples are scripting environments such as Perl, or surrogates like dllhost.exe, svchost.exe and java.exe.
- **Malware**: Malicious or unwanted software that should never be run.
- **Product** – Name of the product.
- **Manufacturer** – Details about the Manufacturer.
- **Description** – Description of the product describing its key functionality.

Processes seen for the first time are by default not approved, and therefore show up in the report for classification by administrators. Additionally, processes that are known to be used for diagnostics or implementing configuration changes to the system are alerted on, and marked as needing approval.

Analyzing a sample report from 126 production servers over a one month period we found 37 systems (29%) ran an overall total of 18 distinct unauthorized processes. The breakdown of these is: three were associated with automatic updates to a runtime environment, seven were desktop applications and data manipulation tools, and eight could not be identified by administrators or security experts in the organization. 76 systems (60%) ran 17 distinct processes that required approval because they can be used for diagnosis or configuration changes.

Report: Change Impact Analysis

Each time a new binary is used on a system this alert shows the application that installed the binary file, and the user context that installed it. LiveOps catches binary installations regardless of whether the installing application participates with RPM, MSI, or PackageAdd. Figure 6 shows the daily alerts generated by iexplorer.exe (web browser) which was used twice to download and install applications. The first time it downloaded and installed msnsearchtoolbarsetup_en-us.exe (MSN Search Toolbar) and the second time it downloaded and installed winamp52_full_emusic-7plus.exe (Winamp Media Player). It shows that Winamp created several binaries on the system, including emusic-7plus.exe (EMusic), which later installed even more binaries.

This report is useful to administrators that may not have expected EMusic to be installed, and might at a later time wonder where, when, and how the EMusic binaries got on their machine. Figure 7 presents a 'drill in' of the detailed process tree at the time of the installations. It clearly shows that the web browser launched the Winamp installation, which in turn launched the EMusic installation.

Identifying the impact that changing a PS entry has on a system's running applications is useful for test verification planning, and for troubleshooting. Knowing the specific PS affected by a change and the

Computer	Impacted App	Changed By App	Additional Information
sysman-21w2k3	Images	iexplore.exe	
Impacted App			
	msnsearchtoolbarsetup_en-us.exe		
	winamp52_full_emusic-7plus.exe		
	First Change (UTC)	Last Change (UTC)	Process Info
	2/28/2006 6:39:12 AM	2/28/2006 6:39:12 AM	1 blocks
sysman-21w2k3	setup.exe	msnsearchtoolbarsetup_en-us.exe	
sysman-21w2k3	Images	winamp52_full_emusic-7plus.exe	
Impacted App			
	pxsetup.exe		
	emusic-7plus.exe		
	winamp.exe		
	winampa.exe		
	winamp52_full_emusic-7plus.exe		
sysman-21w2k3	emusicclient.exe	emusic-7plus.exe	

Figure 6: LiveOps detecting a web browser (iexplorer.exe) downloading and installing two new applications.

affected applications enables administrators to prioritize testing the specific features of the applications controlled by the updated PS. Administrators and support professionals can also significantly reduce the scope of potential root cause PS by knowing which recently modified PS is used by the broken application.

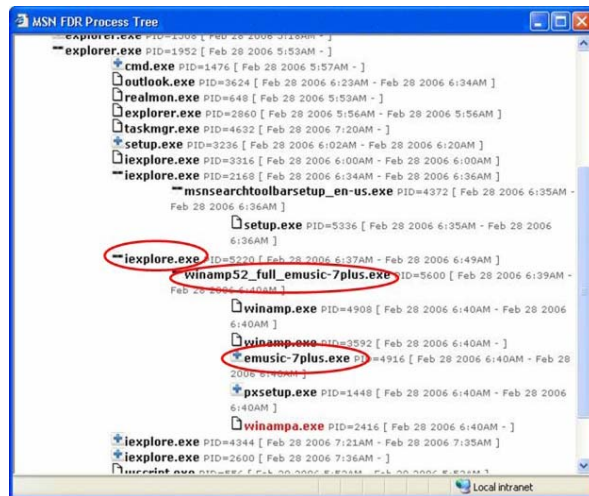


Figure 7: Drill in of the process tree at the time of the installation.

LiveOps identifies the dependencies of each process running on a system by tracking the PS read and modified by each observed process. This list of dependencies, called a manifest, can then be intersected with each change on the system to identify the impacted processes [8, 30]. If the contents of patches or applications are known before they are installed, their impact on existing applications can be predicted by intersecting their contents with the manifests for each observed process. The Application Compatibility Toolkit version 5.0 [19] contains a feature called the Update Impact Analyzer, which uses portions of the LiveOps technology to provide this functionality, enabling administrators to prioritize testing and migration of applications and patches.

Table 2 presents a summary of daily changes affecting the five categories of processes observed across 34 production servers over a one month period.

Surprisingly we see that each day there are several changes made that impact LOB and OS processes, and that each machine on average only receives about half (90 of 163) of the distinct global installations impacting OS applications. This means that the 34 machines are not all receiving the same set of installation changes, whereas the two install changes affecting the desktop category of processes is uniformly applied to all systems. Overall, we see a trend of uniformly applied PS changes to desktop, developer, and home applications. However, less than half of the LOB, management application configuration, and OS configuration changes are consistently applied to all machines. Although we expect systems from all five MSN sites to run the same management processes, and configure them the same, these results show that there are in fact differences.

Report: Sensitive Data

Protecting a company's intellectual property (IP) is of paramount importance because IP is often a critical asset responsible for its competitive advantages. Examples are source code, strategic business plans, and product specifications. Similarly, many customers provide personally identifiable information (PII) to corporations as a part of transacting business. Examples include credit card and social security numbers, contact information, and medical records. Personal Information Protection Laws enacted in 2003 in Japan [13, 14] make companies liable for up to 300,000 yen or six months jail time, for each person affected by PII leaks, which, coupled with an increase in malware and hacking [26] means that corporations need to ensure data is protected.

Further demonstrating the significance of this problem, the CSI FBI report [11] asserts that 59% of corporate security abuse is caused by employees [15]. All anonymous security experts we surveyed on this point thought the number may in fact be closer to 80-90%. Protecting IP and PII from internal employees is challenging because many of them will have legitimate business needs to access the data, and administrators will have implicit access to the data by virtue of managing and backing up information on the server. This means network isolation, firewalls, and

Average Daily PS Change Type		Categories of Distinct Impacted Processes				
		Desktop	Developer	LOB	Management	OS
Across all Machines:	Install	2	182	56	168	25
Per Machine:	Install	2	124	23	95	5
Across all Machines:	Configuration	0	2	4	27	9
Per Machine:	Configuration	0	2	2	10	1
Across all Machines:	Content	0	8	37,063	46	9
Per Machine:	Content	0	8	18,112	22	6
Across all Machines:	Management	0	0	1	59	5
Per Machine:	Management	0	0	1	59	6

Table 2: Impact analysis showing the average daily global and per machine changes impacting each of the six categories of processes observed running across the 34 production servers during a one month period.

	User Setup	Script	Auto Update	Self Update	Developer
Home	7%	7%	51%	35%	0%
Desktop	5%	8%	58%	22%	7%
Lab	2%	5%	61%	32%	0%
Server	1%	17%	38%	44%	0%

Table 3: Distribution of installations by program type for all observed installations.

access control lists (ACLs) will not be effective at solving this problem.

LiveOps addresses this issue by providing an audit report on all files copied to remote network shares and removable devices, for each machine running its agent. The files being copied are then categorized according to their content using a set of configurable classification filters. An alert is then generated for each process instance that contains the user context of the process, the process name, and a list of the sensitive files being made remote.

From examining 383 machine days of logs from 35 datacenter servers and corporate desktops we found: 36 instances of internal documents, six instances of source code, and three instances of applications being copied to network and removable devices. The reports also identified 100's of Corporate IT tools copying logs off of these machines.

Case Study: Tracking Software Ownership

Intuitively, we may think that managing the PS of a system should be easy because executable files and configurations are infrequently created or are modified by well-known software installers. We analyze below our traces to find that installations actually occur quite frequently. Additionally, while most software installers provide manifests, listing the PS entries owned by the installed application to facilitate its clean un-installation, these manifests are often incomplete or incorrect [12]. In the very next section we analyze PS entries across 70 machines and find that many entries cannot be accounted for via the machines' static manifests. To better understand the origins of these orphaned PS entries, we describe point experiments with installing and uninstalling three popular applications.

How Often Is Software Installed?

To identify a software installation we examined our PS traces to identify the creation or modification of files that were later loaded by a process as an executable file. We found that on average 20% of all machine days had at least one installation. However this varied significantly across each environment. 15% of Home and Lab machine days and 30% of desktop machine days had at least one install. Server environments had a wide variance in the frequency of software installations, ranging from 7%-80% of machine days having at least one install. This reflects the variation in change management policy for each Internet service.

While we might have thought that centralized administration of corporate desktops, or Windows auto-update service might cause synchronized updates, this does not appear to be the case. Overall, we observed that most software installations, even in the server environment, occur in an unpredictable manner.

Table 3 describes the distribution of observed installations across install types. We see that processes which update themselves (Self Update), predominantly anti-virus applications, account for a significant portion of installs. Also, enterprise software distribution applications and Windows auto-update account for the majority of software installs in most environments. As we expect, servers have a large portion of scripted installs from administrators manually rolling out upgrades and in-house applications. Installations caused by users running install programs are infrequent. It is interesting to see that our analysis was able to distinguish binary files created and used on developer machines as 'installed' by the developer tools by tracking the processes that created each binary.

Static Software Ownership Manifests

Unfortunately, a statically declared manifest is not always complete. Today's manifests are not always correctly specified, nor do they account for PS created post-installation, such as user preference settings, log files, etc. This means that during software upgrades or removal, entries can become orphaned on the machine. Furthermore, installation or removal of software can fail, or be interrupted which often leaves registry entries and files in an inconsistent PS. Over time the orphaned files and registry entries accumulate on a machine causing a buildup of unused entries that can lead to system problems.¹ Because of this phenomenon, common advice is to reinstall your OS and all applications occasionally to return the system to a known state.

To quantify the significance of this problem, we wrote a tool to extract PS ownership manifests from within the Windows OS. The tool identifies the components installed on the system by analyzing the OS installation configuration files, enumerating the list of programs that have registered with the Windows 'Add/Remove programs' component, the Windows Installer database (WI), the OS configuration for launching applications when a file of a given extension is run, and manifests of patch contents as described in the Microsoft Security Baseline Analyzer

¹Examples can be found at <http://support.microsoft.com/> via the article IDs: 898582, 816598, 239291, 810932, 181008.

		Manifest	Implicit	Data	Temp	Unknown
File	Desktop	18.5%	21.0%	20.6%	8.3%	31.6%
	Server	4.7%	52.6%	2.4%	3.4%	36.9%
	Lab	13.2%	5.8%	9.5%	1.4%	70.1%
Registry	Desktop	28.2%	32.2%	N/A	N/A	39.6%
	Server	10.5%	36.4%	N/A	N/A	53.1%
	Lab	30.3%	31.7%	N/A	N/A	38.0%

Table 4: Average file and registry entries that are specified in manifests, implicitly in manifests, user data, or temp entries. We do not have heuristics to recognize data and temporary registry entries.

tool [20]. We then enumerate all files and registry entries on a machine and compare them with these manifests to identify unaccounted for entries. We associate entries that are descendents of entries that are referenced in the manifests as implicitly associated with the manifest as well. Finally, we filter the remaining list to exclude well known user data files by their extension, and remove entries from well known temporary folders. We define the remaining subset as leaked entries on the system. Table 4 contains the results of running this tool across eight desktops, 20 Server, and 42 lab machines. It shows that 31-70% of files and 38-53% of registry entries could not be accounted for.

To further understand the prevalence of software leaks, we measured the leakage of three popular commercial applications. By running our data collector while installing the application, using it for a short period, and then uninstalling the application; we were able to measure the net increase of file and registry entries on the machine.

The first application was the game ‘Doom3,’ which left nine files and 418 registry entries. The second was the common corporate desktop application suite Microsoft Office, which left no files, but 1490 registry entries. Additionally, it left 129 registry entries for each user that logged into the system and used the program while it was installed. The third example was the enterprise database application Microsoft SQL Server Yukon edition, which leaked 57 files and six registry entries.

Scenario: Managing Abnormal System Activity

Administrators are often required to reactively manage systems that do not operate as expected because of:

1. Hardware problems
2. PS problems such as incorrect configuration or mismatched binary versions
3. Programming logic related issues such as memory corruptions and crashes.

The process for debugging hardware and programming logic related problems is relatively straightforward because there are many tools to aid in the analysis and the correct behavior is known.

For example, solutions often exist for hardware diagnostics such as correctness tests performed in software [41], and often devices like hard-drives are

capable of reporting when they are about to fail [1, 17]. Similarly many tools exist for software debugging such as code analysis tools that are run during the software development process [22], specialized software debuggers [10, 18] and crash analysis software programs for diagnosing application problems when they happen [3, 39].

Configuration related problems on the other hand, are very difficult to debug because tools do not typically exist to analyze the 200,000 settings [36], and 100,000 files [7, 32] that exist on a typical system. Furthermore, the knowledge of which files and settings the application under investigation depends on is typically manually learned by operators as they gain experience with applications. The task of debugging application configuration is further complicated by the complexity caused by frequent updates to configuration, and the customization of system environmental variables and settings.

Several strategies have been proposed for minimizing the potential for configuration problems at the expense of either application functionality or availability. One approach is to statically link libraries with executables to reduce the overall number of executables on the system and thereby minimize the potential for mismatched binary versions. However, doing this makes it more difficult to patch libraries when security vulnerabilities are discovered because all applications using the library need to be rebuilt and reinstalled.

To improve the troubleshooting and forensic investigation of problems, LiveOps provides reports that describe stale PS which identifies applications that are using stale versions of PS compared to the version that exists on disk, the ability to generate ad-hoc forensic reports for specific time ranges that can be filtered by processes, users, and files and setting interactions, and instances of known problems. The remainder of this section describes these reports in more detail and summarizes our results from generating them in MSN datacenters.

Report: Stale Processes

When applications read PS into memory there is the potential for the persisted version of the PS to be updated by another process, causing it to use an old ‘stale’ copy of the PS. An example of this is applying a security patch to a critical file such as the tcpip.sys network driver in Windows, which updates the file on disk.

If the system is not rebooted after the patch has been applied, the old copy of the file is used in memory and therefore the system is still vulnerable to the security exploit. Similarly if an application reads its configuration settings at startup and does not monitor for changes that happen while it is running, any new changes made while the application is running will not take effect. During installations and upgrades it may be expected that some PS will become stale while upgrades are made, however, the length of time applications are stale should be small. We define any process that has not re-read updated PS within five minutes as a stale process.

LiveOps has the unique ability to detect when binary, file, or registry PS read by a process has become stale due to external changes. A report on stale processes is created by correlating all writes to PS with the PS loaded by currently running processes. The LiveOps report contains an entry for each stale process, and enables the administrator to ‘drill in’ to each entry to examine the individual stale PS, showing the time it was modified and the process and user context that made the change.

Figure 8 presents a section of a LiveOps stale report that shows a specific user running Windows Update (update.exe) at 2/28/2006 on three production servers. It shows the exact time that the tcpip.sys driver was modified on each of the machines, and the time when the new version of the driver was reloaded into memory. We can see that the systems were updated between 4:35 am and 5:05 am, but only the first one reloaded the driver six hours later at 10:49 am. This means that the other two machines were still exposed to the security vulnerability caused by the old tcpip.sys binary more than 24 hours later when this report was viewed. Without this LiveOps report the server could have been vulnerable for several days because reboots of servers happen infrequently.

Computer	User	Stale State	Changed By (App)
3 machines			
	User 1	tcpip.sys	update.exe
Computer	Stale Since(UTC)	Stale Until (UTC)	
Machine1	2/28/2006 4:54:59 A	2/28/2006 10:49:07	
Machine2	2/28/2006 5:05:37 A	Now	
Machine3	2/28/2006 4:35:00 A	Now	

Figure 8: LiveOps detected that the updated tcpip.sys binary was not reloaded on the bottom two machines, therefore they are still vulnerable.

We examined several examples of stale processes found from a sample set of 34 machines over a one month period. Several stale OS, Management, and LOB processes were stale for more than 20 hours from software binary updates and management configuration changes. In cases where administrators are troubleshooting problems with these applications, knowing

that they are stale could significantly reduce troubleshooting time, because examination of the newly changed state could mislead administrators to believe that the new values were being used.

The observed stale state ranged from:

1. Cached domain server information which could affect the performance of authentication operations
2. Management configuration which could affect the availability of monitoring information or the ability of administrators to connect and diagnose the system
3. Updated environment variables that could affect LOB applications that required specific path settings to locate needed binaries and data files
4. Web site content and configuration which may be expected to have an immediate affect once modified on the system, or could break other websites due to partially available new content

Query Interface: Forensics and Troubleshooting

The Forensic query interface is useful when administrators want to know who, when or how changes are being made [23, 37, 38]. For example, if malware is detected on the system by anti-malware tools, the forensics interface can be used to identify when it arrived and which user context and process was used to create the malicious files to identify the root cause of the issue and prevent future occurrences.

LiveOps has successfully identified performance problems in several applications that were unnecessarily reading registry entries hundreds of times per second. One instance was found in an LOB application running on a web server where the registry entry \HKLM\SOFTWARE\Microsoft\Cryptography\Defaults\Provider\Microsoft Strong Cryptographic Provider, and all of its values were read 240 times per second. Another case was found in a commercial management product agent that was deployed on a production server, where it was continuously reading all registry entries and values under the \HKLM\System\CurrentControlSet\Services\ key to detect changes in the parameters of daemons installed on the system. The developers of both products were notified and told that they could make their applications vastly more efficient by using the RegNotifyChangeKeyValue function to register for registry changes rather than polling for them.

LiveOps can also be used in an ad-hoc manner when troubleshooting to identify if access violations reading files or registry entries are causing applications to fail, or even for investigating intrusions from hackers where the investigator wants to know which processes were used during the attack, and potentially which sensitive data files were compromised.

Report: Detecting Known Problems

The root cause of configuration problems identified by administrators, support professionals or technically adept end-users, can be used to define assertions

about the correctness of configuration on any system. The LiveOps known problem report is generated by applying codified configuration assertions, called rules, to the registry entry names and values collected from each managed system. The report contains a row for each machine and configuration entry that matches a rule. Administrators can review the rule description that indicates why the entry is suspicious, and view the data contained in the problematic entry.

While several registry scanning products exist, LiveOps is different in that it does not need to run on each managed system to scan the entries because it can instead scan the logs that are centrally uploaded. LiveOps results are also more relevant because the logs contain the registry entries and values that are currently being used by the applications so there are less false positives from entries that are not actually used. Also, user perceived application problems can be correlated with PS usage because the logs contain the times an application used the problematic registry entry. The logs will also contain the exact time, process, and user context that changed the registry entry to the problematic value.

The LiveOps known problem rule set contains approximately 100 assertions created from reviewing the Strider troubleshooter cases [36], PC fragility cases [9], and from critical registry entries identified by MSN administrators such as the setting that controls the operating system paging file. This system can be further enhanced to take advantage of PeerPressure [35] comparisons to identify problematic configurations based on the configuration of similar machines.

Reviewing the known problem report for a sample one month period from 350 desktop, home, and server machines revealed several issues. Most interesting were the 35 servers that had their page file setting configured to null, which caused the system to crash when physical memory was exhausted. Reviewing the problem incidents from the time when these changes were made revealed that several of these servers had in fact crashed from this incident. The reports showed that it was the svchost.exe process configured to support remote registry calls that was used to make the problematic changes, probably from an incorrectly written management script.

Another interesting problem was found on a laptop where the rule for the GSM audio codec configuration detected a missing value. This codec is part of the default Windows installation for playing .wav files. Attempting to play a .wav file on the machine confirmed that it was an actual problem. Resetting the value resolved the issue.

Several of the remaining detected problems related to rules regarding user preference changes that end users may unintentionally make and not know how to undo. For these entries, LiveOps marks the alerts as warnings rather than errors, and if a user from a

managed system complains about an application problem these entries helps quickly resolve related issues.

Scenario: Enforcing Best Practices

Datacenter managers typically define best practices for managing systems to reduce the potential for security vulnerabilities, and to minimize the potential for introducing reliability and availability issues. However, it is a never-ending challenge to audit whether these practices are being followed. As an organization grows the number of its datacenters around the world, it becomes more difficult to educate the administrators on the best practices. This also means that an increased number of administrators will be interacting with the systems, thereby increasing the potential for best practice violations. With more people interacting with systems it is hard to identify which administrators may have inadvertently introduced a problem, and therefore makes it difficult to re-educate them on the best practices to avoid problem reoccurrences.

LiveOps reports have been created to analyze best practices for minimizing exposure to potential security vulnerabilities, and for minimizing potential system instabilities. The initial best practices reports are: Login report which details the users that have been logging in to each server; and Daemons running with local or domain credentials.

Report: Logins

Avoiding unnecessary logins to production servers reduces the potential for hackers to obtain credentials that can be used to connect to other systems in a network. Logging in to a server causes a primary user context to be created, which can be used to connect to remote systems one hop away. If, instead tools are run remotely on the system, a secondary security context that is only valid on the remote machine is used. Furthermore, a best practice for avoiding the potential for problems is to minimize the amount of changes and interactions on managed systems.

A sample of LiveOps login reports for 34 production systems over a one month period shows several examples of logins. Most notably, nine systems were found to be running screen savers 28 times, by detecting instances of the scrnsave.src process. This implies that primary credentials were left on the systems for extended periods of time. It also showed that each system was logged into at least twice, with one system having 11 logins.

Report: Non-System Daemons

Windows systems joined to an active directory have machine accounts associated with them, which can be used as the user context for running daemons. Machine accounts have the same properties as user accounts and can be added to access control lists (ACLs) if needed. It is best practice to run all daemons using the machine account [6] because this will typically not have permissions on remote machines, and

will not require storing user credentials (login names and passwords) locally to run the daemons. Locally stored credentials can potentially be obtained by hackers that infiltrate the system and be used to connect to other systems.

The LiveOps non-system daemon report for 34 machines over a one month period identified several daemons that were not running with the machine account security context. Six distinct processes were identified across all systems, five were management product agents, and one was an LOB process. Each of the 34 systems had violations for at least one daemon running, and a few machines had up to 4. The majority of these daemons were management agents.

Feasibility of Labeling All Changes

The ability to understand all changes made across large numbers of systems is possible only if the rate of new PS is small enough for humans to reasonably comprehend. To evaluate this we determined the first time LiveOps observed process names, PS entries across all managed systems. Using this information we determined:

1. The steady state daily rate of new items observed over an extended period
2. The learning period, which is the number of days taken to observe the majority of items

The learning period determines how long LiveOps must be run before it will reliably generate reports without false positives caused by observing PS for the first time. The steady-state determines how many new PS items must be classified on an ongoing basis for LiveOps to accurately label and understand all observable PS. If labeling is not periodically maintained LiveOps will still accurately and correctly generate all reports, however over time there may be some unknown entries for some reports.

To help prevent this LiveOps provides contextual information that makes it easy to label new PS, and can integrate with other sources descriptive PS information as described earlier. This includes contextual information about the processes using the PS, integration with existing software libraries [24] and work order systems, as well as correlation with previously labeled LiveOps PS. Additionally, automatic inheritance based labeling strategies can be employed such as labeling binaries used by only one process with the process's label.

Figure 9 shows the distribution of newly seen processes from all monitored systems over 39 days since the start of LiveOps data collection. We can see that the learning period is one day, at which time 1000 distinct processes were observed. The steady-state for new processes is typically 0, unless software updates or installations occur. During software updates many processes are created from files created with random temporary file names that do the installation work. There are two solutions to automatically labeling these processes:

1. Using the checksum and size of the process .exe will effectively canonicalize the names of the temporary files into distinct entries across all machines
2. Substring rules can be created to eliminate process names created within known temporary folders.

Figure 10 presents the distribution of newly seen binaries, that is any .dll, .exe, or other executable file that is loaded by any of the monitored machines. Distinct entries are determined by using the checksum stored in the executable header and the size of the binary file. These are the same properties used by kernel debuggers to identify the correct version of symbols to load when debugging a process. We can see that the learning period lasts for one day, when 1400 binaries are first observed, then reaches steady-state where new binaries are only observed if new installations or patches are applied.

The growth of new files and registry entries files since the start of LiveOps data collection is shown in Figure 11, and Figure 12 respectively. For files and registry entries, each day there are a large number of newly generated temporary and cached files with random names. These entries are filtered out in daily counts using simple substring rules that look for known temporary paths in the names. We can see from both graphs that the daily newly observed entries are

Rate of Newly Observed Processes

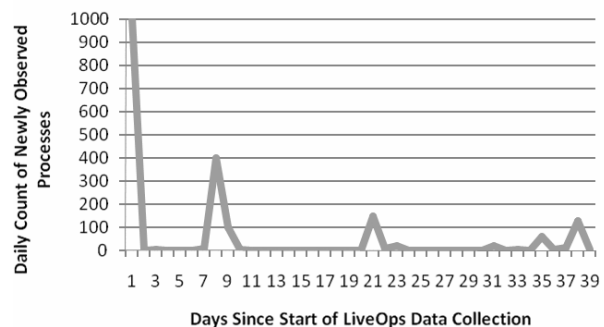


Figure 9: Growth of new processes since the start of LiveOps data collection.

Rate of Newly Observed Binaries

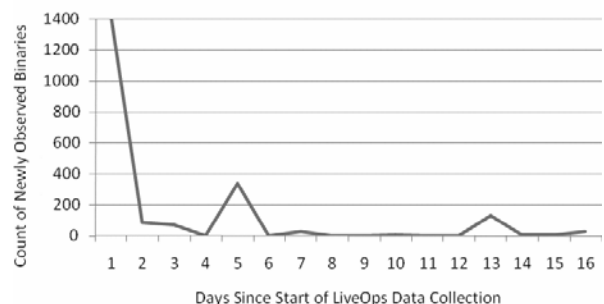


Figure 10: Growth of new binaries since the start of LiveOps data collection.

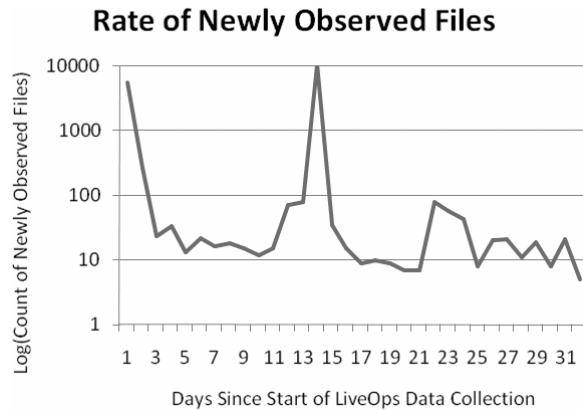


Figure 11: Growth of new files since the start of LiveOps data collection, filtering temp entries.

$O(10^1)$ across all systems, unless new installations or configurations are made, such as those that happened on the 14th and 23rd day. This is a reasonable amount for humans to manually evaluate.

Conclusion

We built LiveOps, a scalable systems management service and comprehensive low-overhead agents, that identify security vulnerabilities, perform compliance auditing, enable forensic investigations, detect patching problems, optimize troubleshooting, and detect malware/intrusions. The service provides a platform for knowledge sharing across organizations and administrative boundaries and allows for seamless integration between analysis results from disparate management products that build on it.

Our analysis of reports from deploying LiveOps in MSN demonstrates how it fulfills a critical need in the configuration management cycle by detecting all system changes, enabling verification of approved requests, and alerting upon unknown modifications. Closing this loop is becoming increasingly important for all datacenters for identifying unwanted changes, and to fulfill auditing requirements. These results illustrate the benefits and feasibility of managing configuration from the OS perspective of interactions between running processes and PS.

LiveOps empowers administrators with new visibility into the relationships between applications and the PS they use. This enables them to more knowledgeably, efficiently, and accurately manage their systems and the applications that run on them.

Author Biographies

Chad Verbowski is an Architect in Microsoft Research. His early academic research on network management translated into a job designing network and systems management infrastructure for MFS Datanet. After surviving the WorldCOM takeover Chad worked at Cisco before joining a management focused software startup company as employee number 5. He eventually arrived at Microsoft where he worked on headless support in Windows 2000, then

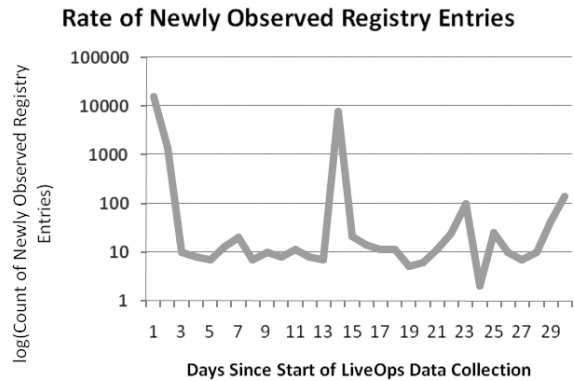


Figure 12: Growth of new registry entries since the start of LiveOps data collection, filtering temp entries.

ran the core development team for the first release of Microsoft Operations Manager before finding his niche at Microsoft Research. At MSR Chad cofounded the Cybersecurity and Systems Management research group, where he focuses on his area of interest: reducing complexity in software.

Juhan Lee is an MSN Architect focused on improving reliability, scalability, and improving operation of MSN's Internet Service Platforms by innovating next generation MS products and research technology into MSN data centers. Juhan joined Microsoft in February 1996 to lead Windows 2000 Datacenter development, Systems core components of Management Server (SMS), and Microsoft Operations Manager 2000. Prior to Microsoft Juhan worked on IBM's CICS/DB2 and OS/2 product lines in Research Triangle Park, and directed distributed middleware and Lotus workflow engine R&D at UNUM Corporation that was ultimately licensed by large software corporations. Juhan enjoys all things electronics and works on home electronics on his free time. He studied Electrical Engineering at North Carolina State University, interned at Nortel and IBM as both Power Engineer and Software Developer where he chose software career by joining IBM in 1988.

Xiaogang Liu is a Software Development Engineer who joined Microsoft in 2005 and has been working on automatic diagnosis and discovery of system anomalies since then. Before that, he was the dev lead of an automatic computer test and configuration project, which has been deployed to two of the top three PC vendors in China and increased the output vastly. His first project in software industry was a Chinese-English dictionary on the Windows platform, where he's responsible for word/phrase lookup under mouse pointer initially and later the main UI. He likes reading books and playing badminton in his spare time.

Yi-Min Wang is a Principal Researcher at Microsoft Research, Redmond, where he manages the Cybersecurity and Systems Management Group and leads the Strider project. Yi-Min received his B.S.

degree from National Taiwan University in 1986. He received his Ph.D. in Electrical and Computer Engineering from University of Illinois at Urbana-Champaign in 1993, worked at AT&T Bell Labs from 1993 to 1997, and joined Microsoft in 1998. His research interests include security, systems management, dependability, home networking, and distributed systems.

Roussi Roussev is finishing his Ph.D. as a student at Florida Institute of Technology. His research interests include distributed systems, security, dependability and program verification.

Bibliography

- [1] Allen, B., "Monitoring Hard Disks with SMART," *LINUX Journal*, 2004, <http://www.linuxjournal.com/article/6983>.
- [2] Arbaugh, W., et al., "Windows of Vulnerability: A Case Study Analysis," *IEEE Computer*, Vol. 33, Num. 12.
- [3] Brodie, M., et al., "Quickly Finding Known Software Problems via Automated Symptom Matching," *ICAC*, Seattle, WA, 2005.
- [4] Brown, A., et al., "Accepting Failure: A Case for Recovery-Oriented Computing (ROC)," *HPTPS*, Asilomar, CA, 2001.
- [5] Burgess, M., et al., "Cfengine: a site configuration engine," *USENIX Computing Systems*, Vol. 8, Num. 3, 1995.
- [6] Chen, S., et al., "A Black-Box Tracing Technique to Identify Causes of Least-Privilege Incompatibilities," *NDSS*, San Diego, CA, 2005.
- [7] Doceur, J., et al., "A Large-Scale Study of File-System Contents," *SIGMETRICS*, Atlanta, GA, 1999.
- [8] Dunagan, J., et al., "Towards a Self-Managing Software Patching Process Using Black-box Persistent-state Manifests," *ICAC*, New York, NY, 2004.
- [9] Ganapathi, A., et al., "Why PCs Are Fragile and What We Can Do About It: A Study of Windows Registry Problems," *ICAC*, Florence, Italy, 2004.
- [10] *The GNU Project Debugger*, <http://www.gnu.org/software/gdb/gdb.html>.
- [11] Gordon, L., et al., *CSI/FBI Computer Crime and Security Survey*, 2004, go.sci.com.
- [12] Hart, J. and J. D'Amelia, "An Analysis of RPM Validation Drift," *Proceedings of the 16th USENIX Conference on Systems Administration*, Berkeley, CA, 2002.
- [13] *Japan's Personal Information Protection Act*, 2003 Law Num. 57, Japan, 2003.
- [14] *Japan's Personal Information Protection Act*, <http://www.privacyinternational.org/survey/phr2003/countries/japan.htm>.
- [15] Jaques, R., *Internal Hackers Pose The Greatest Threat*, 23 Jun 2005, <http://vunet.com>.
- [16] Kim, G. H., "The Design and Implementation of Tripwire: A File System Integrity Checker," *ACM Conference on Computer and Communications Security*, Fairfax, VA, 1994.
- [17] McLean, P., *Information Technology – AT Attachment-3 Interface (ATA-3), X3T13 2008D Revision 7b*, 1997.
- [18] *Microsoft Debugging Tools*, <http://www.microsoft.com/whdc/devtools/debugging>.
- [19] *Microsoft Application Compatibility, Toolkit*, <http://www.microsoft.com/technet/prodtechnol/windows/appcompatibility>.
- [20] *Microsoft Baseline Security Analyzer*, <http://www.microsoft.com/technet/security/tools/mbsahome.mspx>.
- [21] *Microsoft Management Products*, <http://www.microsoft.com/management>.
- [22] *Microsoft Program Analysis Projects*, <http://www.microsoft.com/windows/cse/pa>.
- [23] Moshchuk, A., et al., "Crawler-based Study of Spyware in the Web," *NDSS*, San Diego, CA, 2006.
- [24] *National Software Reference Library*, <http://www.nsr.nist.gov>.
- [25] Oppenheimer, D., et al., "Why do Internet services fail, and what can be done about it?" *USITS*, Seattle, WA, 2003.
- [26] Oswald, E., "Study: Adware Increasing Exponentially," *BetaNews*, September 11, 2006.
- [27] *Payment Card Industry (PCI) Data Security Standard, v1.1*, Sep., 2006, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf.
- [28] Rescorla, E., "Security Holes... Who Cares?" *USENIX Security*, Washington, DC, 2003.
- [29] *Sarbanes-Oxley Act of 2002*, <http://f11.findlaw.com/news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf>.
- [30] Sun, Y., et al., "Global analysis of dynamic library dependencies," *LISA*, San Diego, CA, 2001.
- [31] *Syslog*, <http://en.wikipedia.org/wiki/Syslog>.
- [32] Verbowski, C., et al., "Analyzing Persistent State Interactions to Improve State Management," *SIGMETRICS*, Saint Malo, France, 2006.
- [33] Verbowski, C., et al., "Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management," *OSDI*, Seattle, WA, 2006.
- [34] Wang, H., et al., "Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits," *ACM SIGCOMM*, Portland, OR, 2004.
- [35] Wang, H., et al., "Automatic Misconfiguration Troubleshooting with PeerPressure," *OSDI*, San Francisco, CA, 2004.
- [36] Wang, Y.-M., et al., "STRIDER: A Black-box, State-based Approach to Change and Configuration

- Management and Support,” *LISA*, San Diego, CA, 2003.
- [37] Wang, Y.-M., et al., “Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management,” *LISA*, Atlanta, GA, 2004.
- [38] Wang, Y.-M., et al., “Automated Web Patrol with Strider HoneyMonkeys: Finding Web Sites that Exploit Browser Vulnerabilities,” *NDSS*, San Diego, CA, 2006.
- [39] *Windows Error Reporting*, <http://www.microsoft.com/whdc/maintain/WERHelp.msp>.
- [40] *Windows Event Log*, <http://windowssdk.msdn.microsoft.com/en-us/library/ms732118.aspx>.
- [41] *Windows Memory Diagnostic*, <http://oca.microsoft.com/en/windiag.asp>.

Managing Large Networks of Virtual Machines

Kyrre M Begnum – Oslo University College, Norway

ABSTRACT

As the number of available virtualization tools and their popularity continues to grow, the way in which virtual machines can be managed in a data center is becoming more and more important. A few commercial tools such as VMware ESX and XenEnterprise exist, but they are limited to a certain virtual machine technology and offer no way to expand the tool's capabilities to local needs. This paper discusses an open source management tool, MLN, for large virtual networks transparent of their virtualization platform. The current version supports the two popular open source virtual machine packages: Xen and User Mode Linux. MLN uses an extensible configuration language for the design of the virtual machines and their internal configuration. Large groups of virtual machines can be managed as logical groups. We present a web-server hosting scenario and a on-demand render farm as case studies to show the usefulness of our tool. The text concludes with a short discussion on the difficulties of offering abstraction to virtualization platforms.

Introduction

With the growth in numbers of virtualization platforms, the system administrator will face significant challenges in getting them to work together. Most of the platforms that are popular today have their own areas of application. For example, User-Mode Linux does not require root access to install and can mount folders on the physical host as partitions. Xen has impressive performance and is easy to connect to the LAN. VMware offers graphical front-ends for VM creation and management.

Consider a data center that hosts virtual machines for third parties or a university IT department that provides virtual laboratories for their students.

Ideally, one should be able to choose the virtualization platform that suits the task best. However, the management interface of the platform is currently so tightly connected to the given product that it is impossible to make a decision about the platform without also considering how the virtual machine should be managed.

Although virtualization platforms usually offer a simple management interface, they are often difficult for non-system administrators, e.g., as teachers, to use. Further, these tools offer limited support for specifying attributes inside the virtual machine, such as users or network setup. Once the virtual machine is running, the system needs to be configured additionally by hand, a process that is known to be error prone.

Furthermore, if a user wants to manage many virtual machines, she needs to be able to group virtual machines together in a meaningful way and to manage virtual machines spread out on several servers. The free management tools scale badly because they are aimed at running a few virtual machines on a single host. She must buy a very expensive, more advanced tool in order

to still have control over multiple virtual machine hosting servers. The question of how to manage virtual machines on a large scale is therefore a matter of cost and design tool rather than platform and capability.

Managing VMs also inevitably overlaps with specifying and implementing their configuration. For example, a system administrator is given the task to design and configure a cluster of 50 nodes in a render farm running entirely on virtual machines. A minimal version of the same cluster also has to be build from the same specification for testing. Similarly, a web-hosting company wants to consolidate several customers onto the same physical network using groups of virtual machines that belong to each customer. How can we minimize the drain on the system administrators time resulting from VM administration and configuration? How easy is it to migrate virtual machines between different platforms and/or to migrate them between servers? In general, how can they focus entirely on using the virtual machine rather than on its implementation and configuration?

To summarize, these are the current challenges virtual machine administrators face today:

- Virtual machine management software support only a single virtual machine platform.
- Free management tools are often intended for running few virtual machines. No support for grouping nor the design of larger networks.
- Commercial tools offer better support for larger networks but are proprietary and impossible to modify.
- Host configuration is usually not a part of the management software.

From the field of configuration management we know many tools that group unrelated configuration properties into a abstract configuration language thereby

enabling the user to address the whole computer system or network from a standard interface [1, 2]. They provide an abstraction layer so that the user can focus on the intended policy without knowing the details of the platform nor the necessary steps required to achieve it. Based on the popularity and effectiveness of this approach, we present an open source tool, MLN (Manage Large Networks), that takes a similar tactic to virtual machine configuration and management.

MLN allows the administrator to design, create and manage whole networks of virtual machines and their internal configuration in a template-based fashion. Two popular open source virtualization platforms, Xen and User-Mode Linux, are currently supported. MLN supports an expandable configuration language using plug-ins that allows the data center administrator to include local configurations easily. A MLN network daemon allows for virtual networks to be spread and managed across several physical servers.

MLN is freely available today and has been used successfully as a tool for virtual student laboratories. In this paper we show how the tool can offer significant improvements to the management of virtual machines, despite their underlying platform, in real-life data centers.

This paper is organized as follows: the next section provides some brief background information. We then outline the main features of the configuration language. The two subsequent sections showcase the tool used in real-world contexts: a web hosting scenario and as a management interface for an on-demand render farm. We then present the interaction and control commands of MLN are presented. Finally, we evaluate the tools current capabilities and present directions for future work.

Background

The diversity of today's virtualization platforms make them suitable for a wide range of tasks. We see a growth in the interest of virtual machines in many areas:

- **Consolidation and Commercialization.** Virtualized hosting and service encapsulation is perhaps the most attractive use today, as they are commonly associated with cost saving, flexibility, uptime and security.
- **Research.** A virtual machine is more adaptable in terms of assigned hardware resources and fits well into self-management scenarios [3, 4].
- **Testing.** Creating test-beds for services and software is also becoming more common.
- **Education.** Advanced student labs can be implemented with less cost and more flexibility using virtual laboratories [5].

User-Mode Linux [6] is a version of the GNU/Linux kernel that can be run as an application on a running Linux system. The User-Mode Linux kernel is started on the command line, and host parameters

such as memory size and filesystem image are supplied as arguments. Folders can be mounted as partitions, and one does not require root access to start virtual machine instances. User-Mode Linux is considered to be lightweight in terms of resources and easy to install. A switch emulator (`uml_switch`) is supplied as a tool and enables the user to create network topologies entirely in user space. User-Mode Linux does not offer any higher level configuration tools, although several third party software projects exist today (with varying progression) [7].

Xen is a virtual machine monitor that uses the concept of parallelization [8, 9] to enable several concurrent operating system instances to run simultaneously on a thin management layer called a "hypervisor." Xen virtual machines (called *domains*) have low overhead and are considered to be almost as fast as if the operating system were running directly on the hardware. Xen installation and management requires root access. An attractive feature of Xen is the ability to migrate running virtual instances seamlessly across physical servers without down-time.

Connecting Xen virtual machines together in networks is done using bridge devices on the physical server. A bridge device functions the same way as an Ethernet switch and can either provide isolated internal networks on the server or bridge the physical network, making the virtual machines appear to be regular hosts on the LAN. A Xen domain is defined in a configuration file that addresses virtual machine features such as memory, disk image and simple network parameters. A Xen daemon (`xend`) is responsible for managing the domains. A tool called `xm` will create a single virtual machine based on the supplied domain configuration file. A commercial tool called XenEnterprise is available for purchase which features increased server, management and resource control [9]. From the available information on the XenSource site at the time of this writing it is difficult to assess the management and design capabilities of XenEnterprise.

VMware [10] is a well-known actor in the virtual machine industry. For brevity, we will consider the freely available tools currently offered by VMware and how they fit into our approach. VMware offers a free product called "VMware Server," which has both a web and graphical application interface for managing virtual machines even on remote servers. The software offers an easy way to create a single new virtual machines but has no way to define groups of virtual machines. Also, since every new virtual machine is created graphically it becomes cumbersome if a user wants to design a large network of say 50 virtual machines spread out over 15 physical servers and make sure they have a consistent configuration. A simple tool, called VMware Player offers a quick way for users to run single pre-configured virtual machines. VMware also has a group of products aimed at hosting scenarios, but since they are not freely available, they are not considered in this text.

MLN: A Management Tool for Virtual Machines

MLN (Manage Large Networks) [11] was first used in 2004 as a tool for providing a virtual firewall lab running User-Mode Linux for students [5]. It has since then been expanded to support Xen as a virtualization platform and to include a plug-in framework. MLN can be downloaded from <http://mln.sourceforge.net> and has its own installer, which also downloads and installs a version of User-Mode Linux. Xen must be installed separately.

The MLN configuration language contains both system variables and grouping mechanisms. In MLN, a logical group of virtual machines is defined as a *project*. A file in the MLN configuration language will typically define one project.

Defining Projects

The structure of the language is a hierarchical sequence of blocks containing keyword/value pairs. A block is enclosed in curly brackets ({ }). A keyword is generally expressed in the form keyword value but is not bound to it. Some keywords are lines with several parameters. It is often natural to place one keyword/value per line, but semicolons can be used to place several pairs on the same line.

Each host and network switch will have one block. All hosts in a project do not have to be connected in the same network.

A project has no restrictions regarding the number of hosts or switches. The only mandatory part of a project description is a block of global definitions with at least the name of the project. Project names must be unique for each user. Definitions of one or more hosts and perhaps network switches constitute the network topology. Hosts can have several network interfaces that can be assigned to switches in arbitrary topologies.

Here is an example of a ring-topology:

```
global {
    project ring
}

host router1 {
    network eth0 {
        switch A
    }
    network eth1 {
        switch B
    }
}

host router2 {
    network eth0 {
        switch B
    }
    network eth1 {
        switch C
    }
}
```

```
host router3 {
    network eth0 {
        switch C
    }
    network eth1 {
        switch A
    }
}

switch A { }
switch B { }
switch C { }
```

This is a simple but complete MLN project. In later examples we will show how configurations such as network addresses, users and startup commands are included.

Features for Larger Projects

Language features such as superclasses and variables are helpful when the project is big. We will review these two features next.

Superclasses are a concept from Object Oriented Programming. They describe a class which another class is a subclass of (i.e., a parent). In MLN, a superclass is a description of a virtual machine from which other virtual machines can inherit from. A superclass virtual machine will not be built by MLN. Its most common use is to define a configuration that is to be kept constant and let a group of hosts point to it.

In the example below, the virtual machine node1 inherits all the keywords from the superclass common. It also specifies additional keywords, such as the network interface address. Notice that hosts are free to override keywords from a superclass.

```
superclass common {
    memory 128M
    free_space 1000M
    xen
    network eth0 {
        netmask 255.255.255.0
    }
}

host node1 {
    superclass common
    network eth0 {
        address 10.0.0.1
    }
}
```

Hierarchies of superclasses can be constructed. Hosts only inherit from a single superclass (or superclass hierarchy).

MLN supports string variables in its syntax. This enables the user to keep information consistent across keywords. Consider the following example:

```
global {
    project example1
    $password = 2mf9fmcaioa8w
}
```



```

host node {
    root_password $password
    users {
        jack $password
    }
}

```

The variable `$password` is defined in the global block and used later on inside a host. Variables have scope, so if a variable is used, MLN will look for its value upwards in the block structure and lastly inside the global block. Variables can be expanded into strings if the variable name is enclosed in brackets (`[]`).

Virtual Appliances

Every virtual machine has its own filesystem. One approach to virtual machine management is to create new machines that boot into an installer the first time and install a new version of an operating system. Another approach is to use a ready-made filesystems which are installed and configured with software already.

MLN supports a repository of these filesystems, called *templates*, from which the user can choose from. Templates vary in size based on the amount of installed software they contain. A virtual machine based on a template of this kind is the basis for what is called *virtual appliances* [12] which can be specialized to perform specific tasks (as the examples later will show). The encapsulation of software components in this way has the benefit that experts can put together and properly configure software, and enables users to hit the ground running with a working virtual machine. For example, in educational contexts, this allows students to focus on using a software tool without having to learn how to install and configure it first.

A variety of templates can be downloaded from the MLN web-site. Users and system administrators can also modify existing virtual appliances as well as create new ones.

Plug-ins

It is not the intent of this tool to re-invent configuration management paradigms in its own language. The plug-in architecture is a way to allow other configuration management tools to be integrated as easily as possible with MLN.

A plug-in that is executed can do two actions: access the entire MLN data tree and change the project before it is built, or configure virtual machine filesystems during the build process. Plug-ins have no need to write their own parsing code.

In the following example, we want to build a project where the virtual machines use the configuration management tool `cfengine` [1] for internal maintenance. The template used in this project already has the `cfengine` software installed, but for flexibility, we want to be able to define the `cfagent` policy inside the MLN project as a block inside a host or superclass. The `cfagent` policy should be written to a file `/cfengine/inputs/`

`cfagent.conf` when the project is built. Here is a MLN project with an embedded `cfengine` policy:

```

subclass common {
    template ubuntu-server-cfengine.ext3
    cfagent {
        control:
        any::
            actionsequence =
            (
                shellcommands
                processes
            )
    }
}

host agent {
    cfagent {
        shellcommands:
            "/usr/bin/updatedb"
        processes:
            "cron" signal=hup
    }
}

```

A plug-in in the Perl programming language that writes the above specified `cfagent` into a file does not have to be more than the following code:

```

sub cfenginePlugin_configure {
    my $hostname = $_[0];
    if ( getScalar("/host/$hostname/cfagent")){
        my @cfagent_policy =
            getArray("/host/$hostname/cfagent");
        writeToFile($hostname,
            "/cfengine/inputs/cfagent.conf",
            @cfagent_policy);
    }
}
1;

```

The benefit of this approach is that it is easy to combine MLN with tools that the community is experienced with and that can handle long-term management of the host while it is running. Many system admins have established policies which can be integrated this way using a small amount of code and removes the task of adding the policy manually on each virtual machine. We will see an example later where a plug-in is used to modify the data structure and not the filesystem.

Distributed Projects

Until now, the examples have all been on the same server. MLN also provides a network daemon for distribution of projects among several physical servers. A physical server in MLN called a *service_host* because it provides a hosting service to the virtual machine. A physical host must be made aware of it being a service host in its local MLN configuration files.

A virtual machine is assigned a service host the following way:

```

host startfish {
    service_host huldra.iu.hio.no
}

```

```
memory 96M
network eth0 {
    address dhcp
}
}
```

Project Organization

All the files belonging to a project are stored in a dedicated project folder. The contents of each project folder is the start and stop scripts for the network switches and each VM together with its filesystem image (unless it is placed in a LVM partition). Starting and stopping a project will result in the corresponding scripts being called. UML does not possess any other way to interact with it then through the command line in the time of writing. Xen, on the other hand, is working on an RPC-based approach for VM management which in time might be possible for MLN to interact with.

In the following example, we show a configuration for a data-center that provides virtualized hosting in the form of *virtual sites*. Customers can deploy a gateway and a set of servers on a back-net for their services. A typical example would be a web service with redundant load balanced servers. For simplicity, we omit another tier of database servers.

The physical layout is set up with a single gateway server and a back-net of hosting nodes. The gateway server will host all the virtualized gateway machines. It is possible to physically mirror the gateway server also. A single customer may span one or several of the back-end nodes. Back-end servers may contain one or more virtualized machines from several customers. The customers can chose the amount of web-servers they want to deploy based on the expected load on their web-sites. See Figure 1 for an example setup.

Every server runs the MLN daemon. Each virtual site is represented as one MLN project. The virtual machines in the project are spread across the servers using the `service_host` keyword. The project is built across all the servers that host a node from that project.

```
global {
    $cust_name = kafe
    $default_gateway = 10.0.0.141
    project $cust_name
}

# general settings
superclass common {
    xen
    lvm
    root_password *****
    free_space 1000M
    memory 128M
    term screen
    template ubuntu-server.ext3
    network eth0 {
        bridge back-net
        netmask 255.255.255.0
    }
    files {
        /customers/${cust_name}/www
        /var/www
    }
}

host gw {
    superclass common
    service_host gateway1
    memory 256M
    network eth1 {
        address 128.39.73.101
        netmask 255.255.255.0
        gateway 128.39.73.1
    }
}
```

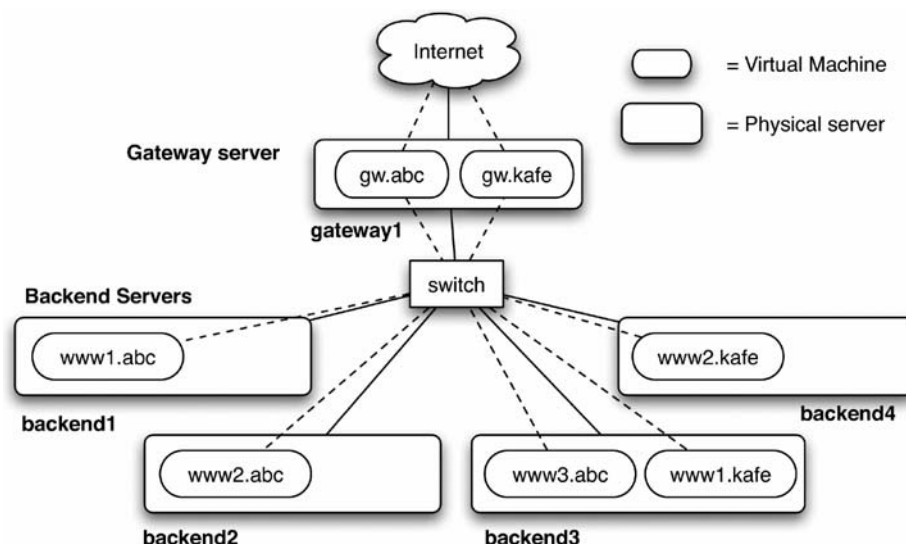


Figure 1: A web-hosting scenario where virtual machines are encapsulations for customer services. Two customers are accommodated in this setup: “abc toys” with three web-servers and a gateway and “kafe on-the-corner” with two web-servers and a gateway. In MLN they are represented as the two projects abc and kafe.

```

network eth0 {
    address $default_gateway
}

startup {
    echo 1 > /proc/sys/net/ipv4/ip_forward
    iptables -t nat -A POSTROUTING
        -o eth1 -j MASQUERADE
}

host www1 {
    superclass common
    service_host backend3
    network eth0 {
        10.0.0.142
        gateway $default_gateway
    }
}

host www2 {
    superclass common
    service_host backend4
    network eth0 {
        10.0.0.143
        gateway $default_gateway
    }
}

... continues to node N ...

```

The example above shows three virtual machines connected together where the host gw has an extra network interface to the outside. The superclass common defines common keywords for all the virtual machines. In one case, the memory keyword is overridden locally by the gateway host. A variable is used to keep track of the gateway address on the back-net. This way we make sure all the back-end nodes point to the correct address and that the gateway actually has that address. The keywords xen and lvm enable the virtual machines to run on the Xen platform and to put their filesystems in LVM partitions for maximum performance. The files block defines what files should be copied into the filesystems at build time. In this case, it is the source files for the web-servers.

The project is spread over three physical servers. The following command can be used to build the project:

```
mln build -f kafe.mln
```

MLN will attempt to contact the daemons on all the involved servers except where the build command is launched. Once the project is built, it can be started using the following command:

```
mln start -p kafe
```

A remaining task is to configure the virtual site to its intended product. Generally this might be done by the owner of the project. Auto-configuration using promise theory and roles on virtualized sites like this was explored in a separate paper [13]. The hosting company can also offer dynamic configurations, where the

number of web-servers is adjusted to the real-time loads on the web-site.

Case Study 2: An On-demand Render Farm

Managing a cluster for rendering can be expensive for small companies as one needs to support a location and hardware for it. In addition, hardware performance increases each year making the local render farm outdated quickly unless one has extra room for expansion.

In this scenario, we consider a small animation company that does not support their own render farm. Instead, they contract with a data center to provide virtual machine hosting for a render farm of virtual machines that the animation company manages themselves. The cost model is pay-per-use, so the animation company only has costs when they are doing actual work, something they consider as an advantage. The data center can rent out their servers to other customers as well and may even run several customer networks at the same time. Moreover, since the data-center is likely to upgrade their servers on a regular basis, they are more likely to attract customers of this kind who will get better performance over time with no extra costs.

The way this is realized with MLN is that the administrator at the animation company has a small, local test-bed on a single machine running a light-weight virtualization platform. There, the template for the render nodes and the master is maintained. Once the animation company has a new contract, a render farm is deployed from these templates. A contract with the data center is made with regard to the number virtual machines to deploy and their resources.

To ease the design of the MLN project for the render farm we introduce a plug-in, *autoenum*, that enumerates the render nodes for us. Here is a project for one master and 50 render nodes:

```

global {
    project renderfarm_customerX
    # the following block contains the
    # configuration for the autoenum
    # plug-in
    autoenum {
        superclass render_node
        addresses enum
        addresses_begin 2
        numhosts 50
        network 10.0.0.0
        service_hosts {
            #include /tmp/servers.txt
        }
    }
    $gateway_address = 10.0.0.1
}

superclass common {
    term screen
    xen
    lvm
}

```

```

superclass render_node {
    superclass common
    template renderNode.ext3
    free_space 1500M
    memory 256M
    network eth0 {
        netmask 255.255.255.0
        gateway $gateway_address
    }
}

host master {
    superclass common
    template renderMaster.ext3
    free_space 5GB
    memory 512M
    network eth0 {
        netmask 255.255.255.0
        address $gateway_address
        bridge cluster-network
    }
    network eth1 {
        netmask 255.255.255.0
        address 128.39.73.102
        gateway 128.39.73.1
    }
}

```

The entire render farm of 51 virtual machines is specified in only 45 lines of code. Actually, the render farm could be increased to 254 without increasing the complexity of the project. We see the use of two superclasses, `common` and `render_node`. The platform specific details are all in the first superclass. Xen is chosen as the virtualization platform with LVM partitions for their hard-disks. A simple test bed of only a few nodes using User-Mode Linux that runs on a laptop could be realized with only minor changes to the file above. This way, the administrator from the animation company could make sure the software on the two templates works as he intends before the full-blown cluster is created and charges start accumulating.

The `autoenum` block in the beginning of the project sets flags for the plug-in. This plug-in has a different intention compared to the one showed in Section 3.4. Upon parsing, the plug-in will fetch the information from the `autoenum` block and use it to create the rest of the virtual machines that make the cluster. This is done by adding them to the data structure before the project is built. This plug-in is therefore not something that expands the configuration of each virtual machine filesystem, but adds design features and logic to MLN based on local needs.

The list of servers where the nodes are spread out is written in a separate file. The `#include` statement is used to read in the contents of that file during the MLN parsing process. The `autoenum` plug-in will assign the render nodes to the servers.

Control Commands and Monitoring

The MLN command builds and starts the virtual machines and networks defined as projects. Here are some examples:

- `mln build -f example.mln`
Build the project from the file `example.mln`.
- `mln -P /my/important/projects start -a`
Start all the projects in the folder `/my/important/projects`.
- `mln status -p mysql-servers -u`
List all the switches and virtual machines belonging to the project `mysql-servers` that are currently running.
- `mln stop -p web-services -w 120`
Stop all the virtual machines belonging to the project `web-services`. Wait 120 seconds for the hosts to shutdown. After the time is elapsed, destroy the remaining ones of the project.

The last example is useful when the physical host is to shut down and has limited time to wait for all of the virtual machines to shut down properly.

Regular users can use MLN without root privileges while using User-Mode Linux as a virtualization platform. Only users with administrator access can start projects that are based on Xen.

The building of a distributed project is started the same way as for other projects. MLN will contact the service hosts for the virtual machines not intended for the local machine and will send them the project for them to do their part.

Upon receiving the build request, the daemons start to build the project in the background and await a subsequent request from the same client asking for the output. Starting and stopping a project will also result in the attempt to contact the other service hosts so that the entire project is managed simultaneously.

MLN will always start the network switches before the virtual machines. The boot order and time to wait between each virtual machine can be specified. Best practice is to avoid a strain on the system by simply letting MLN sleep a few second between each host starts. An example of this, introducing a three second pause, is:

```
mln start -p example -s 3
```

A project is often part of a bigger context on the network or the physical server. Often times one needs to run specific commands on the physical server prior or after the virtual machines have started, such as adding firewall rules or modifying routing information. MLN provides blocks for additions of shell commands so that they are run by MLN at specified points during the starting or stopping of a project.

Modifying Existing Projects

It is not always possible to initially design a project to be optimal for its task. Once the project is running, certain design-time decisions, like memory or

disk-space, might be re-evaluated and have to be adjusted. The problem is often that the project already is in use and cannot be rebuilt from scratch. This problem was encountered several times when running virtual student labs over the course of a semester (five months).

MLN's approach to this problem is to provide an upgrade command, which will read in a new and modified version of the project and try to upgrade it accordingly. Typical modifications are to change the amount of memory, increase the disk size or even add/remove virtual machines from the project. System specific changes, such as adding users, can also be performed this way. For networks which can scale from a software point of view, like web-servers and computing clusters, the upgrade feature can be used to manage the amount of nodes that participate in the cluster at any given time. The modification of a project can be done manually by the system administrator, but recent literature suggests a range of applications for this within self-managing and adaptive systems [13, 4].

A more fundamental change to the virtual machine would be to change its virtualization platform, like going from User-Mode Linux to Xen. To change service host will result in a migration of the virtual machine between two service hosts. The result of this is that one can start with a lightweight User-Mode Linux virtual machine on a regular laptop or workstation, and the virtual machine could later be moved to a more powerful server using MLN where it would be running on the Xen platform with perhaps more memory assigned to it too.

Monitoring

The user can use MLN to collect the status information from all the servers that run the MLN daemon. The information displayed includes how many projects are running, the number of virtual machines, the amount of used memory and how much memory is left from the allowed maximum for that server. This information is useful for monitoring and planning of new projects.

Here, we see the result of the `mln daemon_status` command on the network discussed in Case Study 1. Note that the total number of projects can be misleading, as several servers can have a part of a project and that every part will count as a single project in the summary.

Servers can be put into groups and status can be queried on a per group basis, thereby giving more specialized feedback. One example is to only show the resources on the servers assigned for testing or the ones used in production. Whether or not a project or a certain host is up is also possible through MLN.

Discussion

Successes

Consolidation of several virtual machine technologies into one tool is a new and challenging task. Until now, it seems, the focus for development of virtual machine monitors has been on performance, and carving out a niche. The authors do not see any direct competition between the virtual machine platforms used in this project. In fact, the sum of them is a greater gain. The user should have ability to choose which one to use without affecting the choice of the management interface. Through MLN we have provided one way to design large virtual networks before thinking about the platform it will run on.

MLN creates start and stop scripts for each virtual machine and switch. As a result, any virtual machine technology that is controllable from the command line, would be relatively easy to integrate into MLN. There is no common API to virtualization today. A stronger effort to provide a common API to all the virtual machine technologies would greatly improve the result for projects like this and enable MLN to support even more virtualization platforms by talking to the API directly.

MLN has been tested and used as a commercial hosting tool for over a year, during which it has provided us with much feedback on the needed features for and limitations of current tools. Many features, such as the plug-in framework, have spawned from this exchange. The plug-in framework allows for administrators to add features both in configuration scope as well as logic without re-inventing the wheel.

MLN has become the standard tool for virtual machine management at Oslo University College. It provides the means for massive virtual student laboratories in security classes as well as a virtual appliance tool for student projects. Other institutions, such as University of Linköping in Sweden and Oregon State University in the US have also benefited from it in educational contexts. A Norwegian ISP uses MLN

Server	# Projs	#vms	Mem Used	Mem Ava	Groups
gateway1	2	2	512	768	gateways,xen
backend1	1	1	128	896	backends,xen
backend2	1	1	128	896	backends,xen
backend3	2	2	256	768	backends,xen
backend4	1	1	128	896	backends,xen
Total	7	7	1152	4224	

Table 1: Status information.

today in their R&D department to rapidly create virtual test-beds.

MLN is one of the few freely available tools that offer “cold migration,” where the virtual machine is shut down first and the filesystem is compressed and copied to the new service host. Live migration is supported in Xen but requires the two servers to be on the same LAN and to have the same CPU architecture and concurrent access to the virtual machine’s disk. This is hard to realize transparently to the user as it is bound to a certain platform. Cold migration works in many scenarios where live migration would fail because the servers are of a different architecture and have no concurrent access to the filesystems. Another benefit of this approach is that virtual machines can change other aspects in the migration process. A User-Mode Linux host can migrate into a Xen host with more memory and a different network setup. This is practical for moving test-beds onto more powerful servers of different architecture and to completely separate locations, changing network parameters in the process. All of this is realized using the `mln upgrade` command.

Current Limitations

MLN’s configuration language addresses both hardware attributes of the virtual machine and system configurations. It is therefore not possible to avoid the challenges of host configuration management. Currently, GNU/Debian based templates, such as Ubuntu Linux, are best supported. MLN should ideally be able to support several operating systems let alone support different Linux distributions. However, such concerns are part of the ongoing effort of a large systems configuration management community. The plug-in infrastructure of MLN is one way to invite seasoned configuration management systems and third-parties to handle the lower level tasks. However, some languages might fit better than others into this framework and certain new requirements might surface. This research is in progress and will be discussed in a later publication.

Sufficient monitoring of the virtual machines is a critical feature for data centers. MLN supports status on projects, hosts and globally. Memory usage, the number of virtual machines and projects on each server can be collected as well. This works well for monitoring a project’s status and to see the level of remaining resources on a physical server. However, a usage indicator as to how much CPU and network traffic is related to each virtual machine might further help capacity planning. Xen has tools like `xentop` and `xenmon` [14] that can monitor network, CPU usage and IO operations of their virtual machines. One improvement to MLN would be to expand the plug-in framework to also enable monitoring and management. This would allow the local data center to develop specializations that assist in capacity planning or fault detection, such as a plug-in that finds free IP addresses or logs operations such as starting and stopping.

Another question is whether or not MLN should provide better encapsulation of each project in order to protect them from each other. In User-Mode Linux, this is possible as the virtual machines run as processes and are assigned to users. In Xen, all virtual machines exist in the same “pool” and have no direct ownership. Although the Xen domains are considered to be securely encapsulated, they might still have network access to other virtual machines. One solution is to create virtual switches on each physical server and to connect those with virtual tunnels. This implementation is in progress at this time of writing and will be presented as a plug-in.

Future Directions

The MLN daemon uses IP-based access control for management access. Added features, such as user support for the daemon and finer access control would indeed be a benefit. This way, one could separate access for building a project and the ability to start and stop them.

Interaction with MLN is currently in the form of a configuration language and shell commands. Although the language features improve design and control over large virtual networks, one can investigate other approaches such as graphical design and control tools. Also, adding support for well-known document formats such as XML may enable MLN to play the role of a back-end for higher level tools.

Future work will also look at the improvement of the distributed management aspects of MLN. Scenarios such as management of large and distributed virtual hosting platforms and how to introduce closer monitoring and fail-over are of particular interest.

Conclusion

We have presented an approach to virtual machine administration that lets the user describe the wanted configuration in an understandable declarative language and then build the virtual hosts and networks from it. The virtualization platform is secondary to the configuration interface. A concept of logical groups of virtual machines enables the user to issue management commands to all virtual machines that belong together. Language features such as inheritance from machine superclasses and variable expansion make it possible to consistently describe large networks in just a few lines and to avoid redundant information.

A plug-in architecture lets the user transparently expand the configuration domain of the language to solve their specialized needs. Part of the toolkit is a daemon that allows management of virtual networks that span several physical servers. All of these features have been harnessed to provide a flexible and powerful way to define, create and manage scenarios for data-centers. Two case studies show the usefulness of our approach; a web-hosting facility and a on-demand render farm are realized using simple configurations and local additions to the MLN language.

Author Biography

Kyrre earned his M.Sc. in Computer Science from the University in Oslo. Apart from his studies, Kyrre has worked as a course instructor at a Linux company where he has written and held courses in system administration and Linux. Kyrre started as a full time Ph.D. student in 2003 at the University College of Oslo. His main research areas are anomaly detection, formal modelling of distributed systems and configuration management.

Acknowledgments

The author would like to thank Professor Mark Burgess and John Sechrest for helpful discussions and pointers throughout this work.

Bibliography

- [1] Burgess, M., "Cfengine: a site configuration engine," *USENIX Computing Systems*, Vol 8, 1995.
- [2] Desai, N., A. Lusk, R. Bradshaw, and R. Evard, "Bcfg: A configuration management tool for heterogeneous environments," *IEEE International Conference on Cluster Computing (CLUSTER'03)*, 2003.
- [3] Liu, X., J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered web application using queueing predictor," *10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, 2006.
- [4] Xu, W., X. Zhu, S. Singhal, and Z. Wang, "Predictive control for dynamic resource allocation in enterprise data centers," *10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, 2006.
- [5] Begnum, K., K. Koymans, A. Krap, and J. Sechrest, "Using virtual machines in system and network administration education," *Proceedings of the System Administration and Network Engineering Conference (SANE)*, 2004.
- [6] Dike, J., "A user-mode port of the linux kernel," *Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta*, 2000.
- [7] *The UMLwiki tools page*, 2006, <http://uml.harlowhill.com/index.php/tools>.
- [8] Barham, P., et al., "Xen and the art of virtualization," *SOSP 03*, 2003.
- [9] *The xensource homepage*, 2006, <http://www.xensource.com>.
- [10] *The vmware website*, 2006, <http://www.vmware.com>.
- [11] *The mln project homepage*, 2006, <http://mln.sourceforge.net/>.
- [12] Sapuntzakis, C., D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum, "Virtual appliances for deploying and maintaining software," *Proceedings of the 17th Large Installation Systems Administration Conference, (LISA '03)*, October, 2003.
- [13] Begnum, K., M. Burgess, and J. Sechrest, "Adaptive provisioning using virtual machines and autonomous role-based management," *SELF – Self-adaptability and self-management of context-aware systems, SELF'06*, 2006.
- [14] Gupta, Diwaker, Rob Gardner, and Ludmila Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," 2005.

Directing Change Using Bcfg2

Narayan Desai, Rick Bradshaw, Joey Hagedorn, and Cory Lueninghoener
– Argonne National Laboratory

ABSTRACT

Configuration management tools have become quite adept at representing target configurations at a point in time. While a point-in-time model helps with system configuration tasks, it cannot represent the complete scope of configuration tasks needed to manage a complex environment over time. In this paper, we introduce a mechanism for representing changes over time in target configurations and show how it alleviates several common administrative problems. We discuss the motivating factors, design, and implementation of this approach in Bcfg2. We also describe how this approach can be applied to other tools.

Introduction

Change is unavoidable in today's complex computer networks. Changing user demands, security vulnerabilities, and external resource integration necessitate frequent reconfiguration of local machines. With the size of networks continuing to grow, there is no reason to expect the rate of configuration changes to decrease.

Three factors affect an administrator's ability to keep pace with the constant demand for configuration changes: efficient deployment of changes, control over how and when new changes are used and deployed by the server, and the ability to understand patterns in configuration changes and propagation. Each of these factors is needed in any tool that comprehensively supports change management.

First, the cost of creating and deploying configuration changes must be small, and the deployment of configuration changes need to be controlled. If each change takes several hours of focused administrator time, it cannot be deployed quickly. Similarly, the administrator must be able to control how configuration changes propagate to the environment from the configuration specification. To address this issue, we implemented and deployed Bcfg2 [3]. We presented an account of the deployment process and the resulting efficiency improvements at LISA last year [5]. From these experiences, we feel we have a reasonable solution for this aspect of the problem.

Once changes can be effectively described and deployed, administrators need to be able to represent changes over time. In frequently changing environments, administrators must be able to control when changes go into effect and how they are deployed. Administrators are currently forced to closely monitor configuration systems when performing complex reconfiguration workflows. This process is quite error prone and can cause serious system faults when it goes awry.

With configuration and system changes explicitly described, administrators have access to a wealth of information about the ebb and flow of the configuration.

Through analysis of this data, administrators can characterize the underlying patterns in configuration changes and change propagation information. We believe this information will be of great use to administrators.

In this paper, we discuss our approach to simplifying change management procedures. In particular, we describe the modifications made to Bcfg2 and show how practical situations benefit from this approach. We also detail how other tools can implement similar solutions.

Background and Related Work

Bcfg2 is one of several configuration management tools that provide the user with a declarative interface to system configurations. Declarative tools allow the user to describe the goal configuration state, as opposed to a set of steps that will produce this result. They produce a set of reconfiguration operations that will result in the proper outcome. LCFG [2] was the first system to employ this model; more recently, Puppet [7] has also used it. Each of these tools uses a central specification to describe the desired configuration of a network of machines. Both tools have been used with repositories under version control; however, neither tool actually integrates with version control processes. We believe that revision control techniques could be leveraged for much more than just change logging and roll back.

System administrators have developed several techniques for dealing with change. Many of these approaches are manual. Many complex changes require manual orchestration and are very fault-prone.

Security audit tools have also become popular in the past several years. Tripwire [6] and Aide [1] are both popular auditing tools. These tools have a rigorous approach to detecting system configuration changes but employ a filesystem-based approach to detecting system changes. While this approach helps administrators locate misconfigurations, it does not integrate with an overall model of system configuration.

Some researchers have studied the costs of system administration. In [4], the authors present a cost model of system administration. They found the application of real quantitative data resulted in reinforcement of several intuitive results and found several interesting new patterns. We feel the availability of more quantitative metrics about the configuration process would improve administrators' decision-making and problem-solving abilities.

Approach

The goal of this work is to augment Bcfg2 in order to explicitly represent changes in configuration specification and client configurations over time. Without a notion of time in the system, administrators can interact only with the current state of clients. They can neither analyze past events nor orchestrate future changes. Both of these capabilities are needed if reliable fault recovery mechanisms and scalable administration processes are to be implemented.

As these detailed statistics are collected, a large pool of data accumulates that can be used to better understand long-term trend information and change propagation to clients. This result is a model for the entire life-cycle of systems, including their point-in-time configurations, reconfigurations, and misconfigurations.

In this section, we discuss our approach in detail. We begin by detailing the motivation for this work. Next, to lay a foundation for our solution, we present a high-level overview of the Bcfg2 architecture. We then discuss our implementation in Bcfg2.

Motivations

Configuration management systems seek to efficiently represent a goal for a large group of somewhat similar client systems. Once we felt that we had a good implementation of such a system, it was only natural to extend the representation into the past and future. We were motivated by several operational difficulties. These fell into three main categories:

- Performing reconfiguration workflows of inter-dependent clients
- Enforcing change management policies
- Providing comprehensive audits of past client configuration states

While tasks of these sorts could be represented, pointwise, by using Bcfg2, their implementations were time-intensive and highly fault-prone. We felt that each of these tasks could be greatly simplified with explicit support from Bcfg2.

Bcfg2 Architecture

Bcfg2 is structured as a client/server application with three main parts: the server, the client, and a reporting system.

The Bcfg2 server houses a configuration specification that describes all aspects of configuration for all managed clients. It uses this specification to build per-client configurations when they are requested. It also

provides all network services required by the Bcfg2 client. The configuration specification is stored in a filesystem hierarchy. The server daemon uses FAM to monitor file system changes so as to efficiently cache specification data in memory. The resulting system has low overhead, as it only interacts with the filesystem when modifications require. The server also produces a record of statistics describing clients, including their current states and Bcfg2-related activities.

The Bcfg2 client consumes the client-specific target configuration and performs all operations on client systems. It analyzes the current state of the client system, compares that state with the target configuration, and produces a set of operations that must be performed in order to reconfigure the client system into the target state. Once the Bcfg2 client has performed these tasks, it uploads a set of statistics describing the results of its operations to the server.

The Bcfg2 reporting system postprocesses the client statistics collected by the Bcfg2 server. It produces textual reports, delivered as emails, Web pages, or RSS feeds. These reports are used to display current client conformance with the configuration specification. Administrators can use these reports to repair systems with incorrect configurations or include new client configuration aspects in the specification. While the ability to describe and propagate a desired configuration is independently useful, the reporting system has proven to be the most critical feature offered by Bcfg2. It allows fluid reconciliation of reality with administrative goals, since reality rarely plays along.

Implementation

Addressing these issues required modifications to all three parts of Bcfg2. We discuss each of these in turn, describing how the added functionality improves administrator control and understanding, and interacts with the other parts of the system. In particular, we realized that tracking time stamp information at these three critical points would provide us with the range of functionality we needed. These modifications provided the infrastructure to implement our solutions discussed in the next section.

Bcfg2 Server

We modified the Bcfg2 server to integrate with a configuration repository managed by Subversion [8]. This integration enables the Bcfg2 server to query the repository for the current subversion revision. The current repository revision is included with all client configurations generated by the server. Upon each update to the repository, the revision is updated. We also added a revision log to the server. This log tracks the repository revision used by the server at all times and can be used to determine which revision of the repository was in use at any past time.

This approach has three main benefits. First, administrators can use Subversion to manage their

configuration specification. The benefits of version control are well known and will not be discussed here. Second, the repository revision number provides a discrete set of configuration timesteps that are explicitly tied to repository contents. Third, administrator intent is documented by the revision log. This allows one to determine the desired configuration state of a client at any earlier time.

In practice, changes are made to a separate checkout of the repository and committed to the master repository. The server can then run any revision of the repository, regardless of the current contents of the HEAD branch. The server repository revision is controlled by a discrete utility. This can be used to upgrade or downgrade the server's copy of the repository.

Bcfg2 Client

We modified the Bcfg2 client to process revisions included in configuration specifications. Each revision is associated with all client statistics and uploaded to the server. Because a client can reconfigure at any time, having a change-based discrete time step as a revision number is essential. This approach avoids the need for the retention of any client-side state.

Bcfg2 Reporting Subsystem

We modified the reporting subsystem to retain all configuration statistic records. Previously, it retained only the newest record and the last record in which the client was correctly configured. We also enhanced it to store all statistics with revision information. We currently have a series of reports that summarize this information in basic ways. Over time, we will develop other reports as we find useful views of the data. Data from the reporting system is the basis for all configuration feedback in Bcfg2.

Results

Overall, we have been pleased with the results of this approach. It has exposed a variety of quantitative metrics that we use to better understand the configuration process on our systems. We have used this information and the enhanced control facilities to implement solutions to a number of subtle administrative problems. In this section, we discuss how these facilities lead to dramatic improvements for three broad classes of problems: change orchestration, or the coordination of changes across several systems to achieve a uniform goal; software enforcement of change management procedures; and analysis of past configuration states and changes. Each of these areas was poorly served by previous generations of configuration management tools. For each, we define the problem and provide a concrete example. We describe how the change-based features of Bcfg2 enable the implementation of reliable solutions to each of these issues. Each of these issues can be very time-intensive in large administrative groups or complex environments, so the potential benefit of each is substantial.

Solutions in each of these areas have become part of the daily administrative process. Administrators have found these techniques quite useful and are using them to reach new heights of productivity. Several frustrating tasks have now been automated, so administrators are more contented as well.

Change Orchestration

Frequently, configuration goals require the reconfiguration of several systems in a coherent fashion. These operations are tightly coupled; that is, operations must be performed in the proper sequence and are contingent on the successful completion of other operations. Misordered operations can result in a number of bad outcomes, ranging from transient failures to overall system failures. These workflows are needed in several common situations. When services are changed, clients must be reconfigured to use new services. Moreover, these services may only be used once they have been properly configured. Likewise, all clients must be removed from services prior to their decommission.

In order to automate this process, we have used repository revision numbers to represent states in a finite state machine. Administrators begin by describing all discrete states as individual repository revisions. Administrators then construct an explicit state diagram detailing how the workflow can proceed. Each workflow step consists of two parts. First, the server begins using a particular version of the repository. Then clients are reconfigured using this version. If all clients configure properly, then the system can proceed to the next step in the workflow. If they fail, the server proceeds to the failure result state for this operation. We have written a script that implements this process using data from the reporting system to determine when clients have successfully entered a state. This process can be composed to produce complex series of operations, complete with failure contingencies.

The technique has two important limitations, however. While operations can be chained, administrators must map out all contingencies into discrete states. This process becomes complicated and time consuming in the face of large combinations of failing and succeeding operations. Also, the time spent in any given state is not bounded. That is, success or failure of an operation may be contingent on the actions of a client that is down or not operating properly. To mitigate this issue, administrators can limit per-operation checks to a series of machines that are pertinent to the operation. While this approach is not universally possible, it makes state checks for many operations much simpler. Also, administrators can query a workflow for blocking issues. This technique allows the to locate clients in need of manual intervention.

Change Management

Change management is a set of techniques that ensures changes are performed in a systematic fashion. These policies are useful throughout the change

process, from the initial creation of changes through the testing and activation of these changes. Change management is essential to guarantee the quality and reliability of changes made on production systems. In short, change management controls the conditions in which changes can be legitimately performed.

Change management policies tend to be site-specific. Many of the factors driving these policies are influenced by the reliability guarantees made to users and the ways that particular systems are used. We described initial work implementing change management policies in [5]; however, these processes were largely manual. Our enhancements to Bcfg2 have allowed us to automate most of the processes by implementing automatic enforcement mechanisms for these policies.

In our environment, two configuration management policies are in place. On our core infrastructure, changes are never performed automatically on critical servers. On one of our production clusters, major changes can be made only during a maintenance window, and changes can never be made while user jobs are running. For each of these policies, we describe the software implementations, our practical experiences, and the pitfalls involved.

Manual Change Deployment on Servers

In some cases, there are critical resources that should never be automatically reconfigured. Each of these machines is configured to run the Bcfg2 client in dry-run mode, through Bcfg2 itself. This policy is not hard to implement; however, it is hard to integrate into administrative processes. In many cases, critical systems will not receive configuration updates in a timely fashion because of the increased cost compared with other systems.

Bcfg2's reporting system provides a useful solution to this problem. All clients, even those running in dry-run mode, upload a set of current state statistics. These statistics can tell whether a machine is configured properly, has been configured recently, and has been configured against the most recent version of the configuration specification. Such information may be used to locate hosts that are either out of date or misconfigured. We use this method, in the form of a nagging email, to remind administrators about their critical machines. This approach has worked fairly effectively; administrators are given all of the information they need to address configuration problems on critical systems quickly.

Maintenance Window

Maintenance windows are a planned period in which changes can be made to systems. They are frequently used on production resources. The use of a weekly maintenance window is a requirement on one of our production clusters. This is required because of system policies and the need to maintain software consistency across all nodes that may participate in a single user job.

Our previous implementation of this policy consisted of a choice between two suboptimal solutions. Without fine-grained repository control, administrators were forced to choose between prepopulating the repository with changes and making the changes synchronously during the maintenance window. Both options have serious drawbacks. If changes are made to the repository prior to the maintenance window, they can be consumed prior to the window. That is, clients that run Bcfg2 in dry-run mode will verify against the wrong configuration and will be marked incorrect. This spurious failure can mask legitimate configuration problems. Also, clients that are rebuilt during this interval will be misconfigured, making the configuration inconsistent across the cluster. If changes are made synchronously, administrators are forced to perform all specification updates during the maintenance window. This can be time-consuming and easily result in forgotten changes that cannot be deployed until the next window. Our previous solution was to run the Bcfg2 client in dry-run mode, except for during the maintenance window, when changes could be made. While this produced the correct result, it was unwieldy for administrators.

Correctly addressing this problem requires two capabilities. First, administrators must be able to conveniently queue changes for later use. Once the repository is stored under version control, this operation becomes trivial. Second, the repository must not be changed between maintenance windows. We modified the repository control script described in the Results section to accept a configuration file describing maintenance window times. This script will allow repository changes only during the specified windows. It can be overridden, if circumstances warrant; however, the script prevents simple mistakes from violating change management policies.

Change Analysis

Experienced system administrators have an intuitive sense for how often configuration changes are performed, whether they result in client changes, and whether those client changes have been deployed. While these are useful instincts, they can be based on flawed or incorrect information. Quantitative metrics would provide a more solid foundation for decision making and problem solving.

We have implemented several reports that summarize change information, on both the client and server sides. From these, administrators can better understand rates of specification change, client change, and change propagation and can understand actual patterns present in their systems. We believe the availability of this information will result in sounder change management policies.

We have also implemented reports that construct timelines of configuration operations. These timelines

aid in problem solving. Users often report new failures after a large number of configuration changes. Change summaries are now readily available, enabling administrators to find likely sources of problems. Alternatively, if a system compromise is discovered, configuration logs can be used to determine whether other systems were susceptible at the time of the initial attack. Moreover, this information can be used to determine how long hosts were vulnerable and when they were initially patched.

Conclusions

We believe this approach to be a vital step toward a long-term goal of complete integration of configuration management into the administrative process. Change occurs frequently in most environments; its omission from the configuration management model is a serious oversight that inhibits the development of system configuration tools in a number of beneficial directions.

Status and Future Work

The Bcfg2 codebase is publicly available and is released under the GPL. It is used at a number of sites worldwide. Source code, documentation, papers, presentations, and mailing list information are available at the Bcfg2 Web site [3]. The features discussed in this paper are in use at Argonne currently and should be included in a stable release by early summer. The information exposed by this work is broadly applicable to the analysis of a number of complex issues in system administration. For that reason, it is impossible to predict which uses will bear fruit. We can, however, suggest several interesting possibilities. Static analysis of history information could reveal a number of interesting patterns in system configuration histories. For example, it could locate clients that are frequently misconfigured or remain misconfigured for long periods of time. Similarly, the deployment of critical security updates can be tracked, producing a list of known insecure hosts. All of these factors can be used to produce sophisticated risk assessments.

Similarly, fine-grained control over change deployment provides a powerful infrastructure for tools to build on. Developments in this area will also be guided by site-based needs. This capability is a clear prerequisite for reliable autonomies. We plan to integrate some basic diagnostic functionality into our current generation of scripts to experiment with autonomic principles. In a similar vein, this work allows the implementation of network unit tests. In principle, it could be used to implement deployment regression tests with automatic backout in the case of failures.

Also, our current implementation of configuration workflows have several serious limitations. We plan to experiment with other models to see whether these limitations can be mitigated or eliminated altogether.

Change Support in Other Tools

We believe this approach to be applicable to any declarative configuration management tool. These tools require two modifications in order to implement the full range of functionality described in this paper. However, the first can be implemented without the second. First, the tool must tightly integrate with a repository under revision control. Revision tracking by the server provides the basis for a historical view of the configuration specification. This provides many of the change management benefits described above.

The second step is to include revision information in any statistics collected by the system. This feedback allows the configuration management tool to detect the entry of clients into particular states. Integration with a feedback system is vital to support the change orchestration functionality described above. In the long term, this functionality will be required by autonomies facilities as well.

Author Biographies

Narayan Desai has worked for several years as a systems administrator and developer in the Mathematics and Computer Science Division of Argonne National Laboratory. His primary focus is system software issues, particularly pertaining to system management and parallel systems. He can be reached at desai@mcs.anl.gov.

Rick Bradshaw holds a BS in Computer Science from Edinboro University. He has been a member of the MCS Systems team since 2001, where he aids in maintaining HPC resources, experimental computing resources, and general UNIX infrastructure. He can be reached at bradshaw@mcs.anl.gov.

Joey Hagedorn is a student at the University of Illinois, Champaign-Urbana. When not studying, he works on several software and hardware projects. He can be reached at hagedorn@mcs.anl.gov.

Cory Lueninghoener earned his MS in Computer Science from the University of Nebraska-Lincoln, where he worked with the Research Computing Facility. He now works with the HPC Administrator team at Argonne National Laboratory, currently focusing on Argonne's Teragrid cluster. Cory can be reached at lueningh@mcs.anl.gov.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U. S. Dept. of Energy, under Contract W-31-109-Eng-38.

Bibliography

- [1] *Aide Web site*, <http://www.sourceforge.net/projects/aide/>.

- [2] Anderson, Paul and Alastair Scobie, "Large scale Linux configuration with LCFG," *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, pp. 363-372, October 10-14, 2000.
- [3] *Bcfg2 Web site*, <http://trac.mcs.anl.gov/projects/bcfg2/>.
- [4] Couch, Alva L., Ning Wu, and Hengky Susanto, "Toward a cost model for system administration," *Proceedings of the Nineteenth System Administration Conference (LISA XIX)*, San Diego, CA, December 4-9, 2005.
- [5] Desai, Narayan, Rick Bradshaw, Scott Matott, Sandra Bittner, Susan Coghlan, Rémy Evard, Cory Lueninghoener, Ti Leggett, J. P. Navarro, Gene Rackow, Craig Stacey, and Tisha Stacey, "A case study in configuration management tool deployment," *Proceedings of the Nineteenth System Administration Conference (LISA XIX)*, San Diego, CA, December 4-9, 2005.
- [6] Kim, Gene H., and Eugene H. Spafford, "The design and implementation of tripwire: A file system integrity checker," *ACM Conference on Computer and Communications Security*, pp. 18-29, 1994.
- [7] *Puppet Web site*, <http://reductive-labs.com/projects/puppet>.
- [8] *Subversion Web site*, <http://subversion.tigris.org/>.

NAF: The NetSA Aggregated Flow Tool Suite

Brian Trammell – CERT/NetSA, Carnegie Mellon University
Carrie Gates¹ – CA Labs

ABSTRACT

In this paper we present a new suite of tools – NAF (for NetSA Aggregated Flow) – that accepts network flow data in multiple different formats and flexibly processes it into time-series aggregates represented in an IPFIX-based data format. NAF also supports both unidirectional and bidirectional flow data by matching uniflows into biflows where sufficient information is available. These tools are designed for generic aggregation of flow data with a focus on security applications. They can be used to reduce flow data for long-term storage, summarize it as the first step in numerical analysis, or as a back-end to flow data visualization processes.

Introduction

Many organizations, including universities, private industry, Internet Service Providers (ISPs) and government organizations, are monitoring and storing network traffic information. Due to the overwhelming volume of network traffic, many of these organizations have chosen flow formats over full packet capture or packet header information. This information is then used for a variety of purposes, such as billing or capacity planning, with security monitoring being particularly common.

A number of tools have been produced in the past few years for processing flow data for both network usage and security analysis. For example, OSU FlowTools [4], argus [10] and SiLK [5] each have a significant user base. However, each of these programs produces flow data in a different format, the tools from each are not interoperable, and there are no programs that can perform consistent analysis across each of the different formats.

In this paper we present the NAF (NetSA Aggregated Flow) suite of tools. These tools have been designed for interoperability with a variety of flow collection systems and analysis tools, with a focus on common security analysis tasks using time-series aggregated flow data. To that end, the *nafalize* flow aggregator can read raw flow data from multiple input formats (e.g., SiLK, argus) as well as from data sources supporting the IPFIX [1] standard for flow export. To bridge the gap between unidirectional and bidirectional flow data formats, *nafalize* can match uniflows into biflows if both directions are available. The *nafscii* flow printer converts aggregate flow data into whitespace-delimited text for simple import into a variety of numerical analysis tools.

nafalize was designed to be a general-purpose flow aggregator. It provides a flexible aggregation facility for

generating time-series aggregated flow data, supporting aggregation by any set of flow key fields, and counting of octets, packets, and flows in the aggregate, as well as by distinct source and destination addresses. It includes a facility for sorting aggregated flow data and limiting output to produce time-series Top-N and watch lists. The *nafilter* tool allows further filtering and sorting of this aggregated flow data.

We begin with an overview of related work, focusing on different suites of flow tools and the functionality they provide. We then describe the NAF suite of tools, including the functionality of each of the tools along with a description of the options available. We present details of the design and implementation of the *nafalize* and *nafilter* tools and subsequently provide some usage examples that have been deployed in an operational context. The last section includes concluding comments and plans for future work.

Related Work

The conversion and aggregation of flow data for analysis purposes is certainly not a new area of work, and there exist several other tool sets which address some of the problems the NAF suite was created to address. Most of NAF's functionality can indeed be duplicated by stringing together combinations of these tools; we believe that what NAF adds to the state of the art is the combination of flexible, time-series aggregation and sorting of flow data with multiple input format support in a single, relatively easy to use package. We examine some of these other tools below.

In its default mode of operation, the *ipagcreate* tool from the *ipsundump* [3] suite, maintained by Eddie Kohler at UCLA, works much like *nafalize* and *nafscii* in series, with two important differences. First, *ipagcreate* is primarily designed to read various packet trace formats; “flow-like” input support is limited to Cisco NetFlow summary files and Bro connection summaries. Second, *ipagcreate* has no notion of time series; it is analogous to running NAF with a bin size larger

¹This work was performed while with the CERT Network Situational Awareness Group at Carnegie Mellon University.

than the entire scope of its input; using `ipagcreate` to generate time-series aggregates would require multiple invocations.

The OSU FlowTools [4] `flow-report` tool can produce the same type of aggregate reports, with the same caveat that it does not natively support time series data.

The SiLK [5] flow analysis suite can be used to do many of the aggregation tasks performed by `nafalize` and `nafilter`, though somewhat less easily.² A set of `rwfilter` invocations can be used to filter flows, simulating flow key masking for limited key spaces. The `rwcut` tool provides only simple time binning and aggregation. `nafalize` also supports `biflow` data natively, and unique host and port counting, which are not presently provided by SiLK.

The FlowScan [2] tool, maintained by Dave Plonka from the University of Wisconsin, is another time-series flow analysis environment. Unlike NAF, it is focused specifically on visualization of summary time series flow data, and as such uses the round-robin database provided by RRDTool [13] for data storage and aggregation. However, it does support import of flow data in multiple formats; including CAIDA's `cflowd`, the OSU flow-tools package, QoSient's `argus`, and the Lightweight Flow Accounting Protocol.

NCSA's CANINE [8] tool performs largely the same function as `nafalize`'s first stage, transcoding among a variety of flow formats as a conversion front-end for NCSA's flow data visualization tools. CANINE is also capable of IP address and time sequence anonymization. However, it does not itself provide any support for aggregation of flow data.

Description

The NAF suite presently consists of four tools: `nafalize`, which aggregates flow data into a common aggregated flow data format (the NAF data format, based upon IPFIX); `nafilter`, which sorts and filters NAF data; `nafscii`, which prints NAF data as white-space-delimited ASCII text; and `naflowd`, which loads NAF data into a relational database. The NAF data format itself is handled by `libnaf`, a library installed with the NAF tools; this arrangement allows the easy creation of tools that interoperate with NAF.

All the tools support common options for input and output routing. They can each run as command-line tools or as daemons. In the latter case, the tools can watch directories for input files, processing them as they appear and forwarding the output to other directories; in this manner, chains of daemons can be built to support specific workflows.

`nafalize` reads raw flow data in a variety of formats, filtering, aggregating, and sorting it into the NAF data format. The aggregation function performed

²Indeed, one of the initial motivations behind NAF's creation was to provide an easier method of producing time-series aggregates and unique counts from SiLK data.

by `nafalize` is specified on the `nafalize` command line by an aggregation expression. This expression maps relatively tightly to the design of `nafalize`; that is, each "clause" in the expression corresponds to a particular process within `nafalize`'s data flow. The aggregation expression is described by the following pattern:

```
bin [uniform | start | end] <n>
    (sec | min | hr)
[uniflow] [perimeter <perimeter-ranges>]
[<filter-expression>]
aggregate [sip[/<mask>]] [dip[/<mask>]]
           [sp] [dp] [proto] [sid]
count [hosts] [ports] [total]
       [flows] [packets] [octets]
[<filter-expression>]
[<sort-expression>]
[label <output-label>]
[aggregate ...]
```

Note that multiple aggregate clauses are supported; this is used to specify fanout as in the "Fanout" section in "Stage 3," below. The label phrase serves to differentiate output files in this case.

The first filter expression is applied to the entire data set so that all later processing is performed only on this filtered data (and is described more fully in the "Stage 2" section, below). The second filter expression applies solely to the individual aggregation (see the "Stage 3" section). Thus each individual aggregation can be on data that has been filtered differently. The filter expression itself is described by the following pattern:

```
filter
[time <time-rangelist>]
[sip [not] <ipaddr-rangelist>]
[dip [not] <ipaddr-rangelist>]
[sp [not] <unsigned-rangelist>]
[dp [not] <unsigned-rangelist>]
[proto [not] <unsigned-rangelist>]
[flows <unsigned-rangelist>]
[packets <unsigned-rangelist>]
[octets <unsigned-rangelist>]
[shosts <unsigned-rangelist>]
[dhosts <unsigned-rangelist>]
[sports <unsigned-rangelist>]
[dports <unsigned-rangelist>]
```

The sort expression defines the ordering of records in the output. The output is always sorted in time order first. Within each bin, output records are given in arbitrary order by default. If a sort expression is present, output records within each bin are sorted by the fields given in ascending or descending order. The `limit` phrase limits the output to the specified number of records per time bin; it can be used to generate top-N lists. The sort expression is described by the following pattern:

```
[sort (flows | packets | octets |
       shosts | dhosts | sports |
       dports | sip | sp | dip |
       dp | proto)
      [asc | desc]]
[sort ...]
[limit <n>]
```

`nafilter` reads NAF aggregated flow data, filters and sorts it, writing NAF aggregated flow data. As with `nafalize`, `nafilter`'s operation is specified by a filter/sort expression, which has the same mapping to `nafilter`'s internals as the aggregation expression above. The filter/sort expression is merely a filter expression followed by a sort expression, as defined above.

Each of the clauses in the expressions corresponds to a process in the data flow of each application; see below for more.

`nafcii` is the NAF flow printer. It reads NAF aggregated flow data and writes it out as whitespace-delimited text. It is used for simple aggregated flow display, and for exporting NAF aggregated flow data to other analysis tools (*e.g.*, the R [12] statistical computing and visualization environment) which can handle whitespace-delimited data.

`naflowd` is an online loader for inserting NAF aggregated flow data into a relational database. It was built largely to support the internal workflow of a

specific project at the CERT Network Situational Awareness Group.

Both `nafcii` and `naflowd` are extremely simple in design, reading in NAF aggregate flow records and writing them out after transforming them; they will therefore not be considered in further detail in this paper.

Reference the manual pages in the NAF distribution for detailed usage instructions (available at <http://www.cert.org/netsa/tools/nafe>).

Design and Implementation

In this section, we describe the common data model and storage format these tools use, then explore the design of `nafalize` and `nafilter` in detail.

Data Model

NAF's internal data model is based on time-binned, aggregated, bidirectional flows. Each NAF record represents a collection of flows sharing a given flow key or subkey within a given finite time period

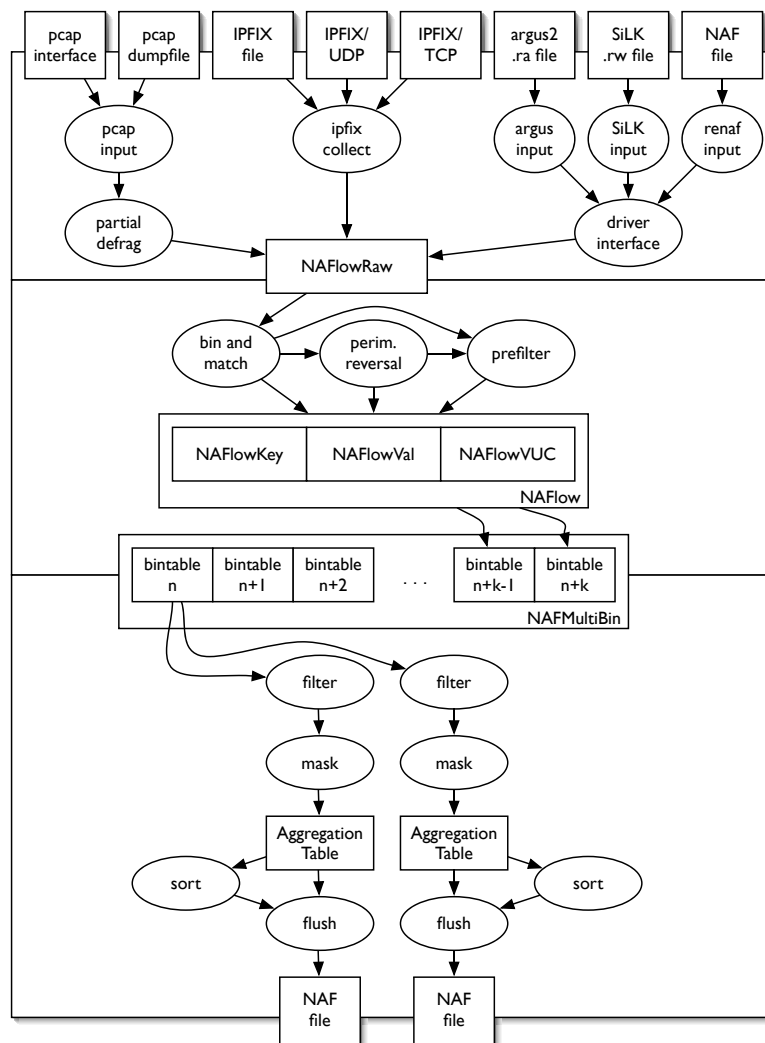


Figure 1: Schematic diagram of `nafalize`.

(bin). Flow keys (type `NAFlowKey` in the NAF source) are composed of some combination of source and destination IP address and prefix length, IP protocol, source and destination port, and source (or sensor) identifier. Flow values (type `NAFlowVal`) are composed of six aggregate counters (for octets, packets, and flows, each in the forward and reverse directions) and four unique value counters (for distinct source and destination IP addresses, and distinct source and destination ports).

NAF uses an IPFIX-based external data format, as described in the IPFIX Protocol Specification [1] and the IPFIX Information Model [11]. Bidirectional flow information is represented as in Biflow Implementation Support in IPFIX [14]. IPFIX was chosen for its self-describing nature; the ability to dynamically define record templates, useful because *nafalize* can produce output with any subset of key and value fields; and for ease of present and future interoperability with IPFIX-based flow metering and collection tools. Each NAF data file is a serialized IPFIX message stream containing the IPFIX templates required to interpret the records contained within the file.

nafalize

nafalize is the NAF flow aggregator. It is an input-driven application, capable of converting a variety of flow formats. It can read flow data from files, function as an IPFIX Collecting Process (reading IPFIX messages over TCP or UDP), or capture packets from an interface via *libpcap*.

nafalize is roughly organized into three stages, with well-defined interfaces between them. These three stages are raw flow input, binning and matching, and aggregation and output; they are described below. A schematic diagram of *nafalize* appears in Figure 1.

It should be noted that presently, all of these stages run in series – for example, when a packet or flow is received from the first stage that causes a new bintable to be enqueued, and a bintable is dequeued, that bintable is aggregated and flushed (multiple times, in case of fanout) before the next packet or flow is processed. This may lead to dropped data when NAF is collecting data from a live interface or via IPFIX over UDP. The strict layering of these stages was chosen for future performance enhancement; by splitting each stage into its own thread for running on its own processor in a multiprocessor system, for example.

Stage 1: Raw Flow Input

nafalize supports three raw flow input types: multi-format flow input from files; IPFIX over TCP, UDP, or serialized streams via *libfixbuf*; and packet capture via *libpcap* [7]. A given *nafalize* invocation can only read input from a single type.

The file input facility includes flow format drivers for reading from QoSient Argus [10] 2.0.6, CERT/

NetSA SiLK [5], and NAF files themselves for re-aggregation. New flow format drivers are relatively simple to add, but at this time require integration into the NAF source code; there is no support for dynamically-loaded flow format drivers.

The IPFIX input facility can read from serialized IPFIX message stream files, and IPFIX messages via UDP or TCP. It is capable of reading both unidirectional and bidirectional flows.

The packet capture facility can read from *pcap* dumpfiles (as produced by *tcpdump -w*), or from a live ethernet or loopback interface via *libpcap*. It does partial fragment reassembly – enough to ensure subsequent fragments of a fragmented datagram are accounted to the correct source and destination UDP or TCP port. When capturing packets, *nafalize* simulates counting TCP “flows” by looking for SYN or SYN+ACK packets. Each packet, SYN or not, is then treated as a separate raw flow for purposes of binning and matching, as below.

Each of these facilities successively reads flow or packet records from their respective sources, and normalizes them into NAF raw flow records (type `NAFlowRaw`), which are then passed to the second stage.

Stage 2: Bin and Match

The second stage of *nafalize* consists of four processes: *binning*, *perimeter reversal*, *prefiltering*, and *matching*. Perimeter reversal and prefiltering are optional, depending on configuration.

Each raw flow is first split into time bins. If a raw flow’s start and end times fit entirely within a single bin, that raw flow is assigned to the bin in which it fits. Otherwise, the flow is assigned to bins according to a user-selectable bin selection strategy. Three of these strategies are presently supported: start, end, and uniform. The “start” strategy bins the raw flow completely into the bin containing the flow’s start time; conversely, the “end” strategy bins the raw flow completely into the bin containing the flow’s end time. The “uniform” bin selection strategy divides the flow’s values equally into each bin covered by the time span between the flow’s start and end time, preserving remainders so that values are robust across re-aggregation.

Note that, as a packet has only a single timestamp, every packet-derived “raw flow” will always fit into a single bin. Therefore, packet capture input has the effect of “start” binning no matter what bin selection strategy is employed.

NAF supports optional perimeter reversal of flows. If the user specifies a network perimeter based on a set of IP address ranges and/or CIDR blocks, all raw flows are conditionally reversed such that addresses external to the perimeter are “source” addresses, and addresses internal to the perimeter are

“destination” addresses. Flows not crossing the perimeter are dropped. This is compatible with the semantics of some security tools, such as snort and QRadar, which typically assign source addresses to the “attacker.” This facility is provided for users of such tools, who are more comfortable thinking of networks in terms of “interior” and “exterior.”

After perimeter reversal, each binned flow is then subjected to an optional prefilter. See the “Filtering” section below for a detailed description of filtering in *nafalize*. Prefiltering is provided for performance improvement. Though each flow can also be filtered after matching, matching requires a flow to be held in memory until it is ready for aggregation. Therefore, early elimination of irrelevant flows before matching will increase *nafalize*’s performance.

The binned flows are then inserted into the multibin (type *NAFMultibin*). This structure is a bin-indexed, time-ordered queue of flow tables (type *NAFBinTable*). The appropriate bin table is accessed by time bin; if no table exists yet for a given bin (because a binned flow is the first one assigned to the given bin), new bin tables are created and inserted at the head of the queue.

The multibin’s queue length is determined by the *horizon*. This horizon h is chosen such that the processing of a flow of start time t enables the assumption that no flows with a start time before $t - h$ will be processed subsequently. When processing raw flow or IPFIX data, this is generally set to the active timeout interval of the flow metering process; for packet capture input, the horizon can be the same as the bin size, keeping only one bin table active at once. This design implies that NAF’s input must be at least roughly sorted by time.

When new bin tables are enqueued at the head, old bin tables expire from the tail. These bin tables are dequeued, and passed on to the third stage.

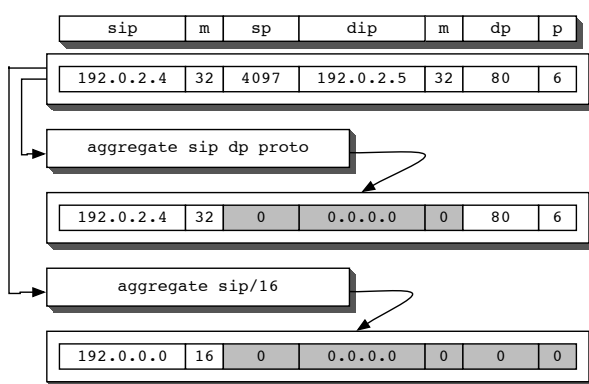


Figure 2: Illustration of mask operation.

Stage 3: Aggregate and Output

The third stage of *nafalize* consists of five processes: *filtering*, *masking*, *aggregation*, *sorting*, and

output. As with prefiltering in the second stage, filtering is optional, and is described in the “Filtering” section below.

Each binned flow that passes the optional filter is masked. Masking consists of projecting each record’s flow key into a subset of the flow key. Subkeys are derived either by dropping fields from the key or by masking IP addresses by a more restrictive prefix length. The masking process is illustrated in Figure 2.

This subkey is then used to create and update aggregate flow records into an aggregation table. The values of each binned flow are added to the corresponding aggregate flow record, and distinct values for flow keys dropped from the subkey (such as hosts or ports) are counted. Once all the binned flows in a bin table have been aggregated, the aggregation table is optionally sorted as described in the “Sorting” section below, then written to the output file.

Fanout

NAF is capable of *fanout*; that is, it can run multiple third stages off a single second stage. Each of these third stages has a different filter and mask, and a separate aggregation table, and writes to a different file. This can lead to significant performance improvement over processing the same data twice, because in many cases the second stage is much more memory- and CPU-intensive than the third.

Filtering

As noted above, *nafalize* may optionally filter raw flows before binning and matching, and binned flows before aggregation. The filtering facility is identical in either case. Each filter is built from a user-supplied filter expression, and contains one or more field specifiers (e.g., source IP, protocol) and ranges of acceptable values for the given field. If a field specifier is present in a filter, then that field must have one of the values in the associated range in order for the flow to pass the filter. Flows that do not pass the filter are simply dropped.

This filtering algorithm does not support arbitrary boolean predicates; instead, it can be viewed as the intersection of a set of unions (or “AND-of-OR”).

When filtering on time ranges, a flow matches a time range if the flow’s start time falls within the time range. For purposes of filtering, the start time of a binned flow is the bin’s start time.

Sorting

As noted above, *nafalize* may optionally sort aggregated flows on output. If a sort expression is supplied by the user for a given aggregation, all aggregated flows in each time bin are placed into an array on output, then sorted using a sort comparator derived from the sort expression. Note that this design constrains the output to be sorted in ascending time order.

The sorting function also supports a limit, which will output only the first N flows per bin in sort order.

In this way `nafalize` can be used to build time-series Top-N lists.

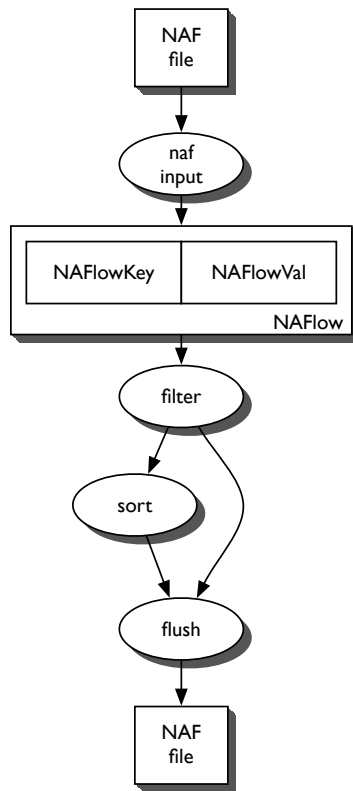


Figure 3: Schematic diagram of `nafilter`.

`nafilter`

`nafilter` is the NAF aggregated flow filter/sorter. Like `nafalize`, it is an input-driven application, though it is limited to only reading files written by `nafalize`. It is roughly organized into two stages: filtering and optional sorting. A schematic is shown in Figure 3.

Filtering operates as described in the “Filtering” section; indeed, the filter implementation is shared between `nafalize` and `nafilter`. It is important to note that the filter operates on raw flows in `nafalize` but on aggregated flows in `nafilter`, so filtering on value fields will have different results in each application. Consider the example of a filter that passes only records with a packet count of one. In `nafalize`, this filter will build aggregates of single-packet raw flows, while in `nafilter`, it will only pass aggregate flows that themselves only have a single packet.

Sorting operates as described above; the sorting implementation is also shared with `nafalize`.

Usage Examples

Here we describe two examples of actual NAF deployments in operational contexts. The first is as part of an internal data collection project at the CERT Network Situational Awareness Group. The second deployment occurred on the network of a large

computing conference as part of a security support effort using SiLK and NAF.

NetSA Preanalysis

NAF is used in the preprocessing of Argus 2.0.6 flow data from a distributed collection infrastructure operated by the Network Situational Awareness group at CERT. The generated aggregate flows support “at-a-glance” visualization and statistical anomaly detection.

Raw Argus flow files (generated by `argus` and preprocessed themselves through `ragator`) are aggregated into four separate labeled files (using the `fanout` feature described below). These files are then loaded via `nafload` into a relational database, from which further analyses are done.³ The `nafalize` command line for this output is:

```

nafalize --daemon --lock --intype argus2
--nextdir argus-cache --faildir naf-fail
--in "argus/*.rag" --out naf-out
bin 5 min
aggregate count flows packets octets
label volume
aggregate count hosts
label talkers
aggregate sip count flows octets
label sources
aggregate dp proto count hosts octets
        packets flows
filter proto 6,17
label pdb
  
```

The four labeled files contain, in 5-minute time series: total flow, packet and octet volume (example `nafscii` output in Figure 4); total distinct source and destination hosts (Figure 5); total flows and octets per source IP address (Figure 6); and total distinct source and destination hosts, flow, packet, and octet counts from per destination port and protocol (Figure 7).⁴

The relational database into which the aggregate flow data is loaded is presently used for two purposes.

First, periodic queries are run against the relational database, and the results are fed into Tobias Oetiker’s `rrdtool` [13] to generate time-series graphs for each of the variables produced (e.g., total data volume per sensor in octets, talkers per sensor, etc.). This provides a simple “dashboard” visualization for the distributed collection system.

Periodic queries are also used to select all the variables available for a given time bin; these are used

³This is not precisely how this works in production. First, the `nafalize` command line is slightly different due to deployment concerns. Second, `nafalize` is run twice for site-specific reasons, with the second `nafalize` instance processing the output of the first instance, using `naf`’s ability to reaggregate its own output. Third, the aggregation expressions in production use the `srcid` field to aggregate data from multiple sensors.

⁴The example data was not generated from production data from the distributed collection system; it is presented as an example of output from the command-line shown.

as independent-variable input into a statistical anomaly detection process based upon Mahalanobis distance [6], which compares each bin to a baseline derived from a larger window of recent aggregate data, and detects time bins which deviate significantly and therefore may benefit from further analysis of the full-flow data. The result of this process is a single time-series “deviance” metric, which is itself fed into rrdtool as above.

Note that we store time-series summary flow data in the relational database, and keep raw flow data in its native (Argus) format for a period of time to permit detailed flow analysis. NAF was deployed in this environment to replace a system which inserted raw flow data into the database for intermediate-term storage, and where all aggregation was done via SQL queries. This system did not scale adequately for our needs; a

more detailed look at the use of relational database technology in raw flow storage is given in [9].

Conference Security

SiLK [5] and NAF⁵ were deployed as part of an effort to provide security support for a large computing conference in late 2005. The example usage and output results presented here were all gathered from this conference. We have represented IP addresses internal to the security conference facility as 192.168.128.0/17, while external addresses have been randomly chosen from 241.0.0.0/8.

NAF is used here as a post-processor for SiLK raw flow data; the per-key binning provided by nafalize

⁵This deployment used an earlier version of the NAF tools which did not support fanout and consequently used a slightly different aggregation expression; the examples have been corrected to use aggregation expressions that will operate with NAF as it is presently available.

date	time	flo	rflo	pkt	rpkt	oct	roct
2006-03-03	13:20:00	363	12	4873	5552	739388	5956007
2006-03-03	13:25:00	279	16	7026	7612	1337665	8042156
2006-03-03	13:30:00	343	11	2599	2208	639824	1616504
2006-03-03	13:35:00	190	9	1355	1077	311763	729521
2006-03-03	13:40:00	223	6	1631	1422	320408	1040939
2006-03-03	13:45:00	223	7	5031	6147	653908	7736319

Figure 4: Argus preprocessing example volume output.

date	time	shosts	dhosts
2006-03-03	13:20:00	35	62
2006-03-03	13:25:00	37	60
2006-03-03	13:30:00	37	70
2006-03-03	13:35:00	31	38
2006-03-03	13:40:00	34	50
2006-03-03	13:45:00	28	48

Figure 5: Argus preprocessing example talkers output.

date	time	sip	flo	rflo	oct	roct
2006-03-03	13:20:00	10.146.0.13	1	0	64	0
2006-03-03	13:20:00	10.146.0.73	27	2	91604	433619
2006-03-03	13:20:00	10.146.0.74	14	0	1436	837
2006-03-03	13:20:00	10.146.0.77	23	0	9286	15266
2006-03-03	13:20:00	10.146.0.82	27	3	7766	5544
2006-03-03	13:20:00	10.146.0.83	14	0	4647	22963
2006-03-03	13:20:00	10.146.0.91	11	0	23202	31724
2006-03-03	13:20:00	10.146.0.95	8	0	3618	35124
2006-03-03	13:20:00	10.146.0.99	2	0	56	0

Figure 6: Argus preprocessing example sources output.

date	time	dp	proto	shosts	dhosts	flo	rflo	pkt	rpkt	oct	roct
2006-03-03	13:20:00	22	6	1	1	4	0	42	47	3910	7894
2006-03-03	13:20:00	80	6	5	15	81	0	1033	1141	107152	1120657
2006-03-03	13:20:00	143	6	1	1	2	0	48	49	2534	34490
2006-03-03	13:20:00	443	6	4	7	40	0	423	431	64404	282673
2006-03-03	13:20:00	445	6	3	1	3	0	3	0	144	0
2006-03-03	13:20:00	993	6	1	1	2	0	4	2	320	266
2006-03-03	13:20:00	53	17	8	6	53	0	91	55	6411	11104
2006-03-03	13:20:00	67	17	5	2	5	0	9	4	3024	1312
2006-03-03	13:20:00	137	17	7	3	9	0	54	0	4203	0
2006-03-03	13:20:00	138	17	5	1	6	0	7	0	1615	0
2006-03-03	13:20:00	5353	17	3	1	6	0	11	0	1365	0

Figure 7: Argus preprocessing example pdb output.

is more convenient to use than the equivalent operations using the SiLK `rw` tools alone, and SiLK did not at the time of this deployment support unique host or port counting as in the third example.

In our first example, we first use the SiLK tools (`rwfilter`) to filter the data to include only those flows that originated from within the internal network, but that were not destined for an internal address. One hour of such TCP data is piped into the following command:

```
nafalize -t silk bin 1 hr
aggregate sip
count hosts | \
nafscii | \
gawk 'if ($1 !~ /date/) { split($3, a, ".");
printf "%d | %d0, a[1]*256*256*256 +
a[2]*256*256 + a[3]*256 + a[4], $4}}'
```

The `nafalize` command aggregates the results from the SiLK commands into one hour bins by source IP address and counts the total number of hosts contacted. These results are converted into ASCII (via `nafscii`). The `gawk` command takes the IP address, which is provided in dotted quad notation, and converts it back to its integer form, printing this value along with the number of hosts contacted by that IP. The results from this command have the following form:

2363326977	1
2363326978	1
2363326980	2
2363327150	32
2363327161	7

These results are then presented graphically (hence the requirement for an integer representation of the addresses) so that a user can gain a sense over time of what was considered normal activity for the network. We present an example graph in Figure 8. This figure indicates five outliers that are potentially worth further investigation.

Figure 9 demonstrates a second command run periodically on the conference network. In order to detect unusual TCP activity, we select TCP traffic that was inbound to the network and that did not originate from within it over a one day period, restricted to only those flows representing failed connection attempts (i.e., where the SYN flag was set but not the ACK flag). This selection is done via the SiLK `rwfilter` command. We again `nafalize` this into one hour bins by source IP address, counting packets and bytes. `nafscii` then converts the output for processing by `gawk` to print the average number of bytes per packet observed, the number of packets, the start date and time for the record and the source IP address in dotted quad. As the selected flows represent failed connection attempts, we would expect the majority of traffic to be 40, 44, 48 or 52-byte single-packet flows. However, here we observe unusual activity. We expect that the cases where there are a large number of packets that average 40 or 44 bytes per packet are indications of scanning activity. For example, the case where IP address 241.194.61.230 has 14,374 packets with an average of 59 bytes per packet might indicate someone who was scanning and trying an exploit against those hosts that responded. This would indicate an IP address whose traffic warrants further investigation.

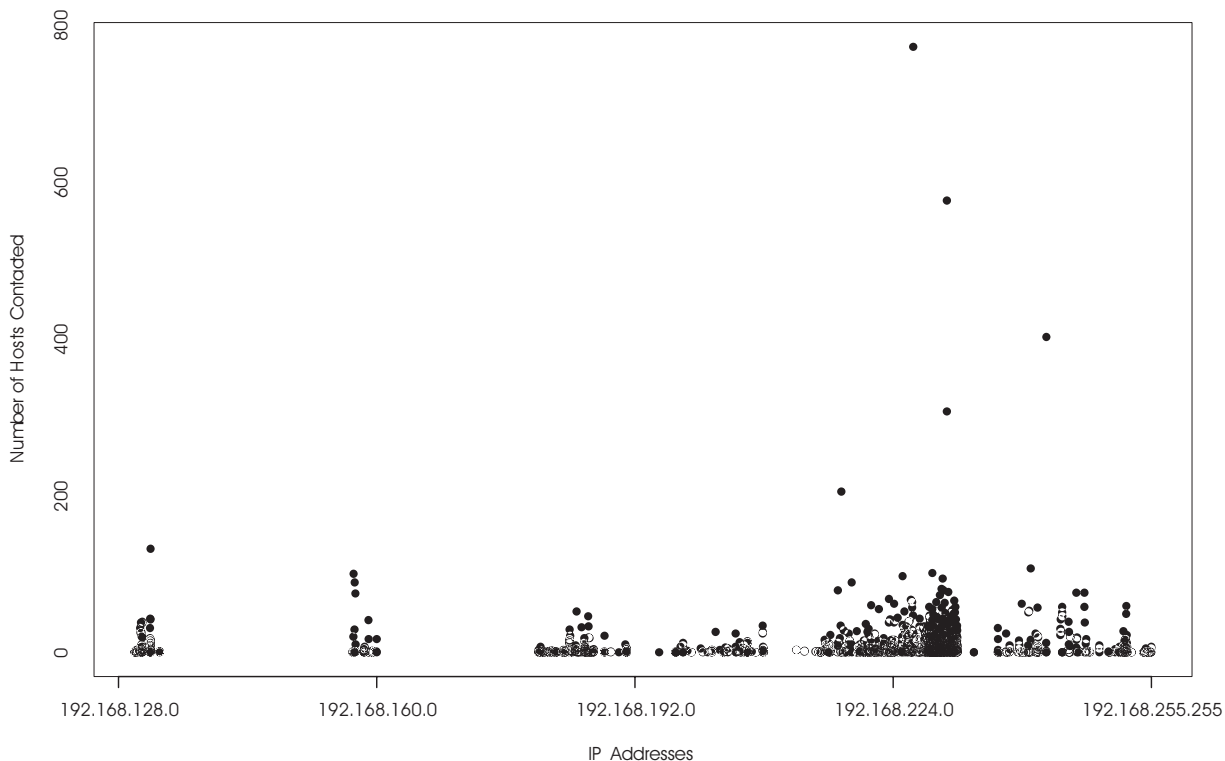


Figure 8: Hosts Contacted Per Hour.

In the third example (Figure 10), we examine all TCP traffic for a single day that is incoming to the conference network and that did not originate from within it. Again we aggregate in one-hour bins by source IP address. We convert to ASCII and process the results, printing out the integer value for the IP address, the number of destination hosts contacted, the start date and time for the record, and the dotted-quad version of the IP address. As there is little reason for an external IP address to connect to a large number of internal IP addresses, this analysis indicates likely scanning activity. Note that IP 241.194.61.230 appears again in this data, lending credence to our hypothesis above regarding their activity.

Our last example demonstrates commands run periodically to detect potentially compromised internal machines. In this case we select TCP flows from an external host to an internal host on port 445, lasting

more than thirty seconds, representing completed connections where more than 1400 bytes were sent. This should extract those hosts that might have compromised the SMB port on a Windows machine. We then use NAF in order to bin on an hourly basis, extracting the start date and hour, the source and destination IP address, the number of flows and the number of bytes. The results from running this command are provided in Figure 11. In this case we find two IP addresses that have potentially compromised a single internal host each.

During the conference, we performed a similar analysis on ports 135 and 22. While much of the traffic destined for port 22 was legitimate, we examined the number of internal IP addresses to which different external IP addresses had connected. We also examined where the external IP addresses were registered, to ensure that they matched known participants rather than likely compromisers.

```
% rwfilter --syn=1 --ack=0 --daddr=192.168.128.0/17 \
--not-saddr=192.168.128.0/17
--pass=stdout --proto=6 --start=2005/11/15 | \
nafalize -t silk bin 1 hr aggregate sip count packets octets | \
nafscii | \
gawk '{if ($1 !~ /date/) printf "%d | %d | %s %s %s \n", \
$5/$4, $4, $1, $2, $3}'
```

40	1	2005-11-15 00:00:00	241.192.13.14
52	2	2005-11-15 00:00:00	241.10.21.189
48	1	2005-11-15 00:00:00	241.11.246.197
44	1929	2005-11-15 00:00:00	241.34.98.164
59	14374	2005-11-15 00:00:00	241.194.61.230
40	23347	2005-11-15 16:00:00	241.192.100.123
740	1	2005-11-15 23:00:00	241.71.1.154

Figure 9: Inbound bytes-per-packet by source.

```
% rwfilter --daddr=192.168.128.0/17 --not-saddr=192.168.128.0/17 \
--pass=stdout --proto=6 --start=2005/11/15 | \
nafalize -t silk bin 1 hr aggregate sip count hosts | \
nafscii | \
gawk '{if ($1 !~ /date/) { \
split($3, a, "."); printf "%d | %d | %s %s %s \n", \
a[1]*256*256*256 + a[2]*256*256 + a[3]*256 + a[4], $4, $1, $2, $3}}'
```

1000079635	1	2005-11-15 21:00:00	241.156.1.19
1006778434	2	2005-11-15 23:00:00	241.2.56.66
1022911850	3	2005-11-15 03:00:00	241.248.101.106
3514611848	283	2005-11-15 20:00:00	241.124.184.136
3703717350	14509	2005-11-15 00:00:00	241.194.61.230
1022903300	15881	2005-11-15 16:00:00	241.248.68.4

Figure 10: Inbound destination host count by source.

```
% rwfilter --bytes=1400-999999999 --dur=30-3600 --dport=445 --ack=1 \
--flags-initial=S/SA --start=2005/11/15 --proto=6 --pass=stdout \
--not-saddr=192.168.128.0/17 --daddr=192.168.128.0/17 | \
nafalize -t silk bin 1 hr aggregate sip dip count total flows octets | \
nafscii | \
gawk '{if ($1 !~ /date/) { if ($6 > 10000) \
printf "445 | %s %s | %s %s | %d | %d\n", $1, $2, $3, $4, $5, $6}}'
```

445	2005-11-15 10:00:00	241.146.88.212	192.168.190.233	18	1209855
445	2005-11-15 11:00:00	241.146.88.212	192.168.190.233	1	44824
445	2005-11-15 16:00:00	241.214.211.244	192.168.190.248	3	231372

Figure 11: Inbound potential SMB compromise detection.

Conclusions and Future Work

We have introduced a new flow aggregation tool suite, the focus of which is interoperability with multiple flow sensor technologies and the reduction of flow data for network security analysis purposes. These tools are designed to be reasonably generic, and apply to a wide variety of analysis tasks. We have explored the design of two of these tools in detail, and provided examples of their usage in two real-world applications.

NAF is under continuing active development at the CERT Network Situational Awareness Group. We have planned several enhancements for the tool suite that will appear in future releases:

NAF's internal data model and aggregation operations presently only support flows with IPv4 addresses; we plan to add support for IPv6 addresses, as well.

To support NAF's use in data sharing applications, future releases of the tool suite will include support for data anonymization, when the aggregation operations do not discard sufficient information to meet an organization's dissemination policy needs. Indeed, the extent to which aggregation operations obfuscate data needs to be better quantified; this is an area for future research.

The use of a single data format at the core of the NAF tools' design also makes it reasonably easy to build new consumers for that data; currently planned is a NAF-to-SVG "printer" analogous to *nafscii* or *naflow*. This would allow the generation of time-series graphs from NAF data without the present need to convert the data into *rrdtool* round-robin databases.

Likewise, the layered architecture of *naflowize* eases the addition of new types of flow input to the tool. Additional flow input drivers will continue to be added to the distribution on an "on-demand" basis.

Furthermore, as NAF's output format is based upon IPFIX, it will be reasonably simple to add support for using *naflowize* as an IPFIX Exporting Process; that is, to allow it to send output over the IPFIX Protocol. When combined with existing Collecting Process support, *naflowize* will be deployable as a "drop-in" aggregating IPFIX proxy.

Present experience with *naflowize* suggests that its performance is bound by I/O (how fast records can be read from disk or the network) and the size of the active flow table. While the performance is obviously dependent on both the data and the aggregation expressions used, *naflowize* run on a stock Dell 1850 tends to process between 100k and 250k records per second with between 5k and 10k concurrent flows in the second-stage *NAFMultiBin*. We plan on performing detailed profiling and optimization in order to increase this throughput.

One performance enhancement suggested by *naflowize*'s three-stage design would be to split the stages into separate threads. This may increase throughput on multicore hardware, but more importantly, it will isolate

output delay from input processing; especially important in the aggregating IPFIX proxy case above.

The *NAFBinTable* is presently limited to the size of available memory. *naflowize* may be extensible to work with extremely large datasets by replacing the underlying *binTable* implementation with one that can utilize on-disk storage when necessary, sacrificing performance for flexibility as needed.

Author Biographies

Brian Trammell is the Engineering Technical Lead at CERT Network Situational Awareness Group. He designs, builds and maintains software systems for the collection and analysis of security-relevant measurement data for large-scale networks. He is also an active contributor to internet-measurement related working groups in the Internet Engineering Task Force. He received his bachelor's degree in computer science in 2000 from the Georgia Institute of Technology, where he also worked as the UNIX systems administrator for the School of Civil Engineering for three years. He can be reached at bht@cert.org.

Carrie Gates is a Research Staff Member with CA Labs where she performs research in enterprise security. She received her Ph.D. in May, 2006, from Dalhousie University. While completing her dissertation, she spent three years working with CERT Network Situational Awareness at Carnegie Mellon University doing security research for large-scale networks. Previous to her doctoral studies, Carrie was a systems administrator for six years. She can be reached at carrie.gates@ca.com.

Bibliography

- [1] Claise, B., *IPFIX Protocol Specification*, (work in progress), June, 2006, Internet-Draft draft-ietf-ipfix-proto-22.
- [2] Plonka, Dave, "FlowScan: A network traffic flow reporting and visualization tool," *Proceedings of the 14th Large Installation Systems Administration Conference (LISA 2000)*, New Orleans, Louisiana, USENIX Organization, pp. 305-317, December, 2000.
- [3] Kohler, Eddie, *ipsumdump and ipaggcreate*, 2006, <http://www.cs.ucla.edu/~kohler/ipsumdump/>, (Last Visited: 11 May 2006).
- [4] Fullmer, M., and S. Romig, "The OSU flow-tools package and Cisco Netflow logs," *Proceedings of the 14th Systems Administration Conference (LISA 2000)*, New Orleans, Louisiana, USENIX, pp. 291-303, December, 2000.
- [5] Gates, C., M. Collins, M. Duggan, A. Kompanek, and M. Thomas, "More NetFlow tools: For performance and security," *Proceedings of the 18th Large Installation Systems Administration Conference (LISA 2004)*, Atlanta, Georgia, USENIX, pp. 121-132, November, 2004.

- [6] Lazarevic, A., L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," *Proceedings of SIAM International Conference on Data Mining*, San Francisco, California, Society for Industrial and Applied Mathematics, May, 2003.
- [7] LBNL Network Research Group, *TCPDUMP public repository*, 2005, <http://www.tcpdump.org>, (Last Visited: 9 May 2006).
- [8] Luo, K., Y. Li, A. Slagell, and W. Yurcik, "CANINE: A NetFlows converter/anonymizer tool for format interoperability and secure sharing," *FloCon 2005: Proceedings*, Pittsburgh, Pennsylvania, CERT Network Situational Awareness Group, September, 2005, <http://www.cert.org/flocon/2005/proceedings.html>.
- [9] Navarro, J.-P., B. Nickless, and L. Winkler, "Combining Cisco NetFlow exports with relational database technology for usage statistics, intrusion detection and network forensics," *Proceedings of the 14th Large Installation Systems Administration Conference (LISA 2000)*, New Orleans, Louisiana, USENIX, pp. 285-290, December, 2000.
- [10] QoSient, LLC, *argus: network audit record generation and utilization system*, 2004, <http://www.qosient.com/argus/>, (Last Visited: 9 May 2006).
- [11] Quittek, J., S. Bryant, B. Claise, and J. Meyer, *Information Model for IP Flow Information Export*, June, 2006, Internet-Draft draft-ietf-ipfix-info-12 (work in progress).
- [12] R Project, *The R Project for Statistical Computing*, 2006, <http://www.r-project.org>, (Last Visited: 12 May 2006).
- [13] Oetiker, Tobias, *RRDtool*, 2006, <http://oss.oetiker.ch/rrdtool/>, (Last Visited: 11 May 2006).
- [14] Trammell, B., and E. Boschi, *Bidirectional Flow Export using IPFIX*, August, 2006, Internet-Draft draft-ietf-ipfix-biflow-00 (work in progress).

Interactive Network Management Visualization with SVG and AJAX

Athanasios Douitsis and Dimitrios Kalogeras
– National Technical University of Athens, Greece

ABSTRACT

The area of visualization has always been one of the most attractive sections of network management technology. Successful management tools must not only fulfill objective management needs but also be aesthetically appealing. Consequently, the seemingly mundane task of presenting information to the user has become almost a true art. The application proposed in this paper is a vehicle for the presentation of network management data using interactive graphs. By using the Scalable Vector Graphics markup language (SVG) [1] and Asynchronous Javascript and XML (AJAX) [2], it strives to aid the rapid development of visually impressive management applications that are accessible through the use of a web browser. These highly interactive and versatile applications can respond to user actions and present data which is organized into layers and is retrieved and refreshed on demand. The data of these layers is generated by network management tools that plug in to the application through the use of a modular framework.

Motivation and Problem Statement

The Need for Abstract Visualization

Visualization of data has always been important in many areas of human knowledge and engineering, as it allows people to perceive information in more efficient ways which, in turn, can expedite the learning process and help understand and deal with relevant problems more efficiently. Many network management tools resort to various forms of visualization to depict the topology of a computer network and the relevant information concerning it. Unfortunately, representation of additional information is often cumbersome due to inefficient visual network topology abstraction.

Furthermore, most management tools tend to use inappropriate ways of producing their visual presentations, resulting in increased development effort and doubtful results from the user perspective. This is because most graphics generation libraries and APIs are too much low-level, which increases the burden of implementing the visualization part and consequently fail to abstract the production of visually rich network representations.

The Need for Interactivity

It is also generally recognized that a large part of the usefulness of a network management tool lies in its interactivity. The element of interactivity is essentially defined as the ability to dynamically change graphical representations and respond to user actions. In fact, interactivity is the very element that generally makes the difference between a real usable application and a mere depiction. Again, the development difficulty lies in the fact that the implementation of interactivity incorporates a great deal of unnecessary details and is generally cumbersome. So, a framework that can help

easily produce highly interactive visual network management applications, will naturally contribute heavily to its success from both the perspective of the developers and the users.

Underlying Technologies and Dominant Trends on Network Management Tools

Depiction Of Networks As Graphs

Visualization of networks has been an area directly connected with the visualization of graphs. This is mainly because of the similarity between computer networks and graphs. In fact, it can be argued that one of the most efficient ways to present information on the status of a computer network to humans, is through a two-dimensional graph where the elements of the network (like routers and switches) are depicted as vertices and the connections between them as edges. Essentially all relevant network management tools today use this approach in one way or another, to provide a representation of real computer networks. Clearly, depiction of networks as graphs is the most intuitive choice for most purposes.

Separation of Functions

One other dominant trend among network management tools is the distinction between the presentation layer and the instrumentation layer. This distinction is highly beneficial as it allows to focus on the individual problems and solve them separately. One added benefit is that, if the model of presentation is well thought out, an interface mechanism can be created which can be potentially reused in many different network management tools. This is relevant to establishing a well defined presentation mechanism to be used by software to build interfaces easily.

Application Interface Technology

A question that has been posed many times in the past is which approach is the most suitable for applications to build a rich client interface, which is also connected with the element of interactivity. It should be noted that this question not only concerned network management tools, but applications of any kind. During the late nineties, it was thought that rich clients that were specifically developed for their distinct application, were the way to go. But with the proliferation of the Web, people soon realized that it is in fact more efficient (and easy) to develop applications that present their interface through a browser. During the last years, with the advent of Asynchronous Javascript and XML (AJAX) techniques, it was proved that web based applications can provide the same experience like any native rich client application. A large portion of the applications of tomorrow will be web based.

Regardless of the way the application interface is generated, in order to be used it must be made available to the user. Among popular network monitoring applications, some of the most common approaches to transporting the user interface to the user are:

1. The client application code is actually run in the server and the user interface is transported to the user terminal by using a specialized client-server protocol such as the X-Window System [3], VNC [4], ICA [5], RDP [6], Sun-Ray [7], etc. This method is very powerful because it can transport a window or an entire desktop anywhere, but its use has declined because it places high loads on the server, it has several security disadvantages, it is cumbersome to deploy and use (special clients, permissions, etc.) and it may have gargantuan bandwidth requirements in order to operate in a speedy manner.
2. The client application is presented through a web browser, which in turn loads a presentation and management Java applet [8] that connects to the server using some special management protocol. This method is very flexible because the presentation and interface capabilities of the Java applets are really powerful and the protocol is specially designed for this particular application. However, the usage of a special client-server protocol for this purpose has other implications. On the server side, the protocol handler will typically have to be implemented from scratch, which introduces additional costs and adds security concerns (from having yet another protocol). Even more, on the client side the Java applet will have to carry classes that implement that special protocol, a fact that increases its size, makes it difficult to extend and more susceptible to bugs.
3. The client application is merely a web browser and the graphical images are produced on the

server side. In this case, interactivity is very difficult to implement. On the server side, it requires the production of multiple images, which places additional burden on the servers. On the client side, it requires elaborate techniques using image maps which are difficult to produce and manipulate.

Generation of Graphics

To address the design requirements that are relevant to the graphical representation of graphs, a powerful way to create graphics inside a web browser is required. Usually, when a custom graphical image must be provided to the user through an application, the image is typically generated by the server on the fly using appropriate graphical libraries (like GD [9]) and fetched to the user. This approach is successfully used in our Multicast Weathermap [10] to generate images of network traffic over a geographical map. However, in many problems, the requirements for interactivity and dynamic manipulation makes the usage of a server generated static images inappropriate. It could be argued that, although server side graphics today constitute the vast majority of solutions today, there is clear indication that this may change in the future.

Solutions where the graphics are generated on the client side and can be manipulated dynamically include VML [11], Adobe Flash [12], Java applets and SVG. Of these options, the usage of SVG is the most appealing from a development perspective as:

- SVG is a W3C specification and the markup language is still being actively developed through new revisions.
- SVG uses an event model and interface to its DOM similar to ordinary web applications, with which there is already great familiarity among developers.
- There are already many browser based implementations available, like:
 - The Adobe SVG browser plugin [13].
 - Opera's implementation embedded inside the Opera browser [14].
 - Mozilla's implementation embedded inside the Mozilla Firefox browser [15].
 - The Apache Batik [16] project which can be used through Java Web Start [17] by a browser that launches the application. This is more cumbersome and is very rarely used.

Modularity

Lastly, today's modern networks are far more complex than yesterday's, and include a host of diverse technologies such as Multicast, IPv6, MPLS, etc., each one with its own requirements and peculiarities. It is unlikely that a giant monolithic management tool can be created which can be on top of all these aspects of the network. The present diversity of network management tools today can only serve as a proof to that assumption. In many cases, it is helpful

to use a modular approach, where many tools exist, each one is specialized in its own specific domain, and they are all using a unified presentation layer.

Design and Architecture

Design Goals

Based on the facts presented so far, the creation of a browser based tool that can be used as a generalized presentation layer for network management applications seems like a solution that can prove quite useful. The design goals outlining the characteristics of such a tool are:

- Usage of the tool through a web browser. As mentioned earlier, the delivery of the application through the use of a web browser is a practice that rapidly becomes commonplace, which easily justifies this choice.
- Representation of rich visual representations of graphs, allowing arbitrary usage of colors, styles, images and other artifacts.
- Graphical representations that are dynamically modifiable during application execution.
- Definition of arbitrary rules of interaction of the graphical elements with the user. This will allow customization to the point where the tool becomes truly intuitive to use.
- Extensibility. Individual presentations may require special capabilities which (at least in theory) should be relatively easy to implement with minimal modification.
- Well defined and simple API. This will make it easy to create new network management tools

using the presentation application.

- Speed. The resulting application should be as light as possible (both on the client and on the server side).
- Security is also of great importance. Unauthorized usage should not be able to easily compromise the security of the system.

General Overview

In our proposed architecture (see Figure 1), the Network Management Station (NMS) collects data from the managed nodes in the network, processes it and makes it available for remote clients to connect and retrieve it. In this client-server approach, the client application is actually an SVG enabled web browser that similarly connects to the server periodically or on-demand and retrieves information server. The server on the other hand is the NMS which will typically do the collection from the managed nodes using protocols such as SNMP [18]. That way, administrative access to the network is assigned only to the NMS while the graphical rendering will be delegated completely to the client browser. Use of the client can of course be carried out from virtually anywhere.

The client application itself is an SVG document that contains code but is otherwise devoid of actual graphic content. To launch the application, the user typically has to navigate to the specific URL of the NMS server where this SVG document resides. The Javascript [19] code that is embedded inside the document is actually the implementation of the network management presentation application. Its purpose is, upon certain events, to retrieve management data from

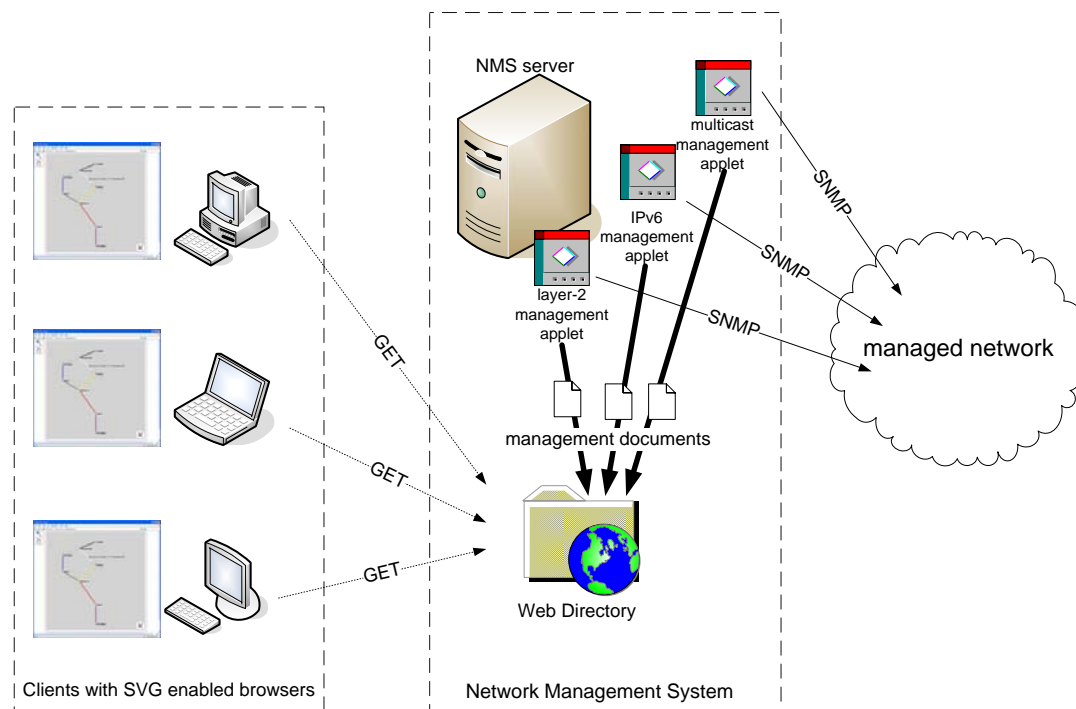


Figure 1: Architectural overview.

the NMS server and accordingly create, modify or delete graphical elements inside a page. The creation, deletion and modification of the SVG elements are done by directly accessing the SVG DOM. Furthermore, the events to which the application is designed to respond to, are:

1. Initial loading of the document. When the user first launches the application by navigating to its URL, the SVG document is empty. Responding to the event of its creation the application must retrieve all required data and create the initial graph.
2. Periodic updates. To make sure that the picture presented to the user is up to date, the application periodically updates the network graph based on fresh data.
3. User interaction. Responding to events generated by the user, the application may have to retrieve additional data and make the appropriate modifications to the graph.

If the content used to draw the graphical elements of the network was to be included inside the initial SVG document, the server would obviously have to dynamically generate this content and send it to the user. Although this is the most common solution, we believe that by supplying an initial document without graphical content and having it populate itself by querying the NMS is a much more elegant solution that can additionally scale up pretty well. That way, the client document that implements the client application is completely static, meaning that it does not need to be generated dynamically at all. It just simply needs to be placed somewhere where the client browser can get it.

Management Data Hierarchy

The management data that must be retrieved by the application is organized into XML documents that are made available by the NMS server to the client applications through HTTP. That way, the client applications can retrieve data by using the XMLHttpRequest [20] class, which is the typical way in which interactive AJAX applications work. The XML management data documents belong to following categories:

1. The **configuration document** that describes the general application configuration, mainly the HTTP locations of the other categories of XML documents to which actual data is contained. This is the equivalent of a configuration file for a conventional application. Upon its retrieval, it instructs the application to create multiple visualization overlays (described later) and supplies the required information to create and draw them. The configuration document is structured in such a way that a hierarchy between the visualization overlays can be described. This hierarchy has a double purpose, on the one hand to define the relative z-axis order of rendering of the overlays and on the other hand to present the user with a structured choice of management layers.

2. Documents that describe the nodes and their positions. Those documents are denoted as **node topology description** documents. A node topology description document usually contains all the managed nodes of the network (routers, switches, etc.) and their coordinates inside the SVG page. Especially for large numbers of nodes, the topographical layout of the nodes can be computationally intensive and cannot be carried out on the client side. Instead, the layout is transmitted to the client through this document along with the catalog of managed nodes. The coordinates of each node are always defined on the NMS server side and cannot be altered by the client user. The NMS server administrator can define the coordinates of each node by hand, which can be very helpful in cases where the nodes are relatively simple and must be drawn on top of a geographical map image. Alternatively, a program which is specialized in graph layout, like GraphViz [21], can be used to fully automate the computation of node coordinates in cases where a large number of nodes exist. A typical case where this is useful, is for large networks whose layout is complex and changes frequently, like a large campus LAN.

3. Documents that describe actual network management data and are denoted as management overlay documents. Each one of these documents refers to a specific domain of management information, for example there may be a document that provides the topology, status and traffic information of multicast inside a network, while another document may provide IPv6 traffic information, etc. Since the coordinates of each network node are already decided by the node topology description document, the management overlay documents define arrows, labels and other graphical elements that use the predefined coordinates of the nodes to be laid out. For example, for a network that is comprised of routers A and B, an overlay document can contain an element that instructs the client application to draw an arrow representing a link pointing from A to B. Obviously the overlay document need not bother itself with the definition and coordinates of A and B as these have been taken care of previously by the node topology description document. Overlay documents also contain interaction information as will be explained later.

It is also possible that some management overlay documents can be loaded on-demand, in response to user interaction. The pointers to the locations of these overlays are provided indirectly by other overlays. For example, when the user hovers over a network link in the graph of a specific overlay, the application may

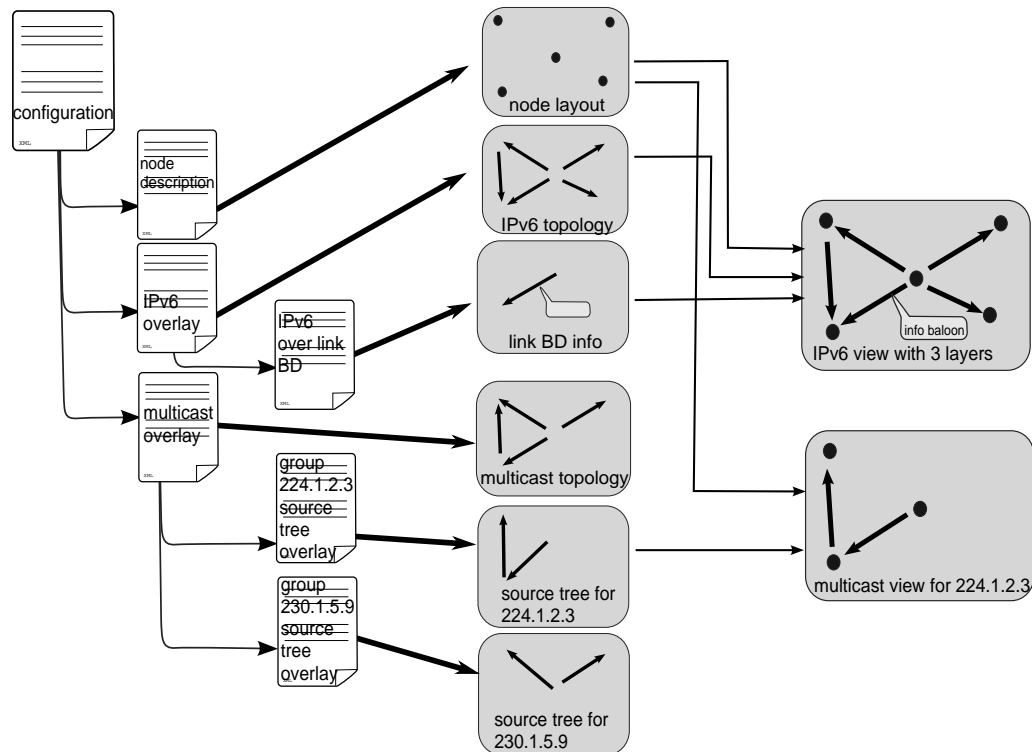


Figure 2: Conceptual arrangement of overlays and dependence to management data documents. Overlays are transposed to produce the final picture.

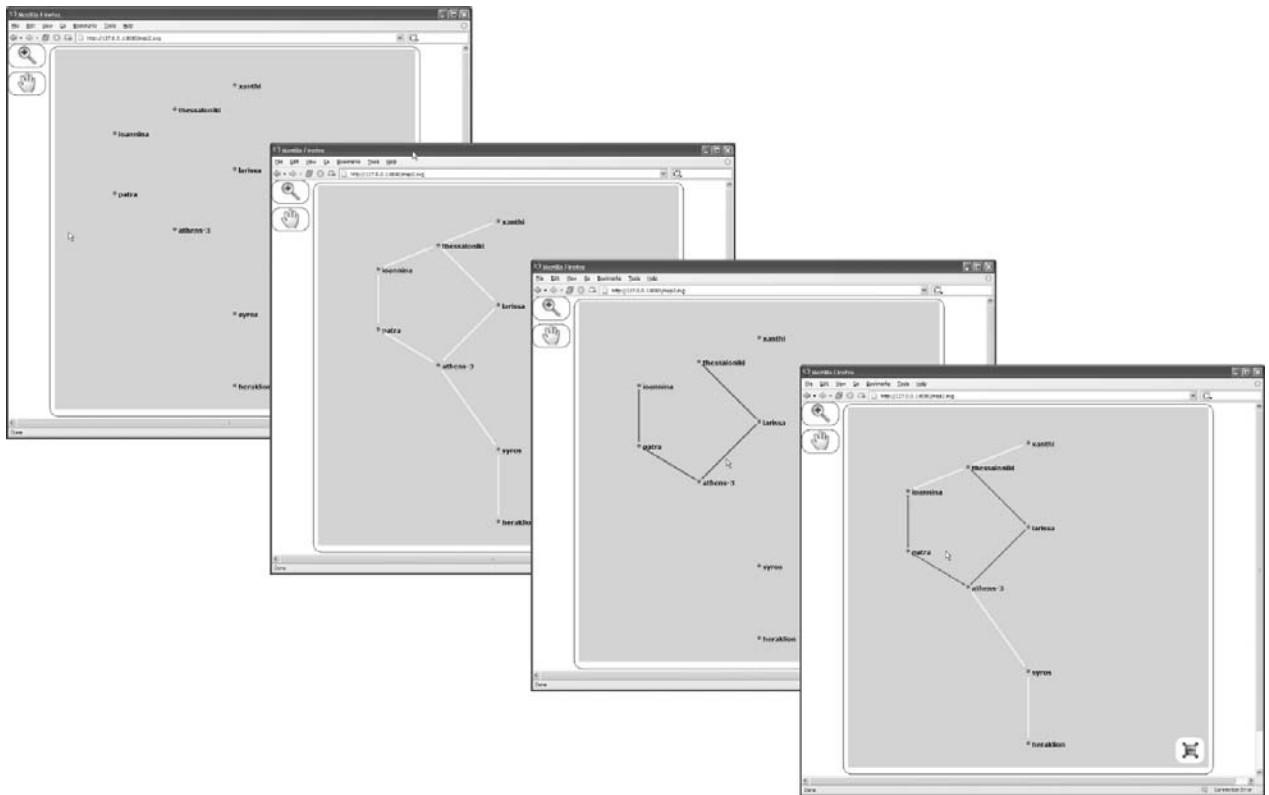


Figure 3: Juxtaposition of 3 different overlays to produce a composite map. The final visualization on the right is produced by combining the other three overlays.

retrieve one or more of these documents and present more information (e.g., which multicast groups are flowing in each direction of the link) to the user.

A conceptual explanation of management data documents and their relation with the presented overlays can be seen in Figure 2. An actual example can be seen in Figure 3.

Management Applets

The main idea behind the selection of the overlay hierarchy is that there are potentially many specialized management tools (management applets) in the server side, and each one of those applets is providing at least one management overlay. The requirements for any applet to plug into the framework are:

The applet must be registered in a configuration document so that clients will be able to know its existence. The registration can be carried out by hand by the administrator when he wishes to plug the new applet to the presentation system.

1. The applet must generate at least one management overlay document, so that a full map overlay can be created. A URL pointing to this document is provided to the clients through the configuration document.
2. The applet may generate as many additional management overlay documents as needed. The URLs for these documents can be typically provided inside other previously loaded management overlay documents.

For example, an applet providing the IPv6 topology of a managed network could be considered. This applet can be written in any suitable programming language and will be invoked periodically inside the NMS. Each time it is executed, it will discover the topology of the IPv6 enabled network and generate a management overlay document that describes it precisely. Management overlay documents can also be created which will be revealing more information about each network link or node. All these documents will be typically placed inside the web document directory of the server to be available for retrieval by the clients.

As long as these applets output XML documents suitable for usage by the clients, the administrator is able to easily create new ones which present new sets of data. The schemas of the XML document categories that were described in the previous list were deliberately crafted to be relatively simple, in order to allow easy manipulation and composition.

```
conf = new XMLHttpRequest(); //create the object
conf.open("GET", "conf.xml", true); //get the document
conf.onreadystatechange = function() {
  if (conf.readyState == 4) {
    ...
    code handling the document inserted here
    ...
  }
}
```

Example 1: Typical code for retrieving the configuration document.

Although this is not a strict requirement, it is proposed that all applets have access to a managed entity registry which is essentially a database of all the discovered managed nodes and other useful information about them. When an applet needs to communicate with a specific node, it may need information such as its IP address, its type, its SNMP community string, etc. These values can be supplied separately to each applet by the administrator using configuration files, but it is much more convenient to have a type of common registry so that all applets can access the same set of configuration. Of course, applets will be able to add newly discovered nodes to this registry. Typically, applets that discover various topologies will also add new nodes.

As implied in the previous list, the graph(s) that depict the various network functions (each one supported by an applet in the server side) must exist together inside a single SVG image. Each one of the graphs is organized into an SVG overlay, the visibility of which can be turned on and off at will by the client user. As the visibility of a SVG element effects the delivery of user interaction events to itself, only the overlays that are visible will respond to user actions. The client code will also seek to periodically update only those overlays that are visible to the user. With this strategy, it is assured that only information that the user actually wishes to see gets retrieved by the client application.

Implementation

Overview

The implementation of the system consists of the client application code that does the graph rendering and the server applets that carry out the data collection and XML documents creation.

The client application is based on Javascript, which is the embedded scripting language in all modern browsers. The application code has a double purpose. Its main function is to collect the data from the NMS server and create the SVG graphics, while its secondary assignment is to handle all the user interaction events and maintain the graphics accordingly.

Collection of Data Using XMLHttpRequest

Collection of data is essentially carried out by retrieving the appropriate XML documents from the server. As indicated, retrieval is based on the XMLHttpRequest class. Similarly with many other web applications, the XMLHttpRequest class is used here by issuing a call that registers a handler.

Example 1 shows typical code for retrieving the configuration document.

This handler will be invoked when the request completes successfully (readyState reaches 4) and its task is to interpret the document that was retrieved. Each of the three schemas of documents (see categories the previous sections) is handled by a specific handler. As they interpret these documents, the handlers will manipulate the graphical content of the SVG page. The application code is programmed to issue XMLHttpRequest.open calls whenever needed, at the application start, periodically or after user interaction.

The programmer has the convenience of treating the XMLHttpRequest object as an XML file or a file of arbitrary format. Using the object as XML is actually preferable, because this circumvents the need to parse the document, a work that would have to be done if it was treated as an arbitrary file.

It is reasonable to expect a lot of complexity to be hidden inside the code that reads the XML structure and creates piecewise the graphical content such as the network graph and other artifacts. Although this is true, there is not too much interdependency between the various steps that are taken to complete this task. So, modification and extension in the future will be easy. Essentially, the whole process of converting a management overlay document into a graphical overlay will be assigned to a method that will be taking the document URI as its argument and will be returning the overlay to be placed inside the main SVG image.

Manipulation of Graphical Content Using the SVG DOM

Manipulation of graphical content is done by using the SVG Document Object Model (DOM). This is also on par with the way other modern web applications operate. The DOM is an object oriented mapping of the structure of a document using classes that map directly onto document elements.

This way, a convenient interface to the content is available to programmatically modify, create or delete

whatever aspect of the page. When a new element (for example, a polygon) must be created, an object of appropriate class is instantiated and then attached to a suitable preexisting object inside the document. The SVG specification uses the grouping (G) element heavily to group other real graphical elements together. For example, to create a circle, the programmer would write a function like that in Example 2, and then use it to generate a circle and attach it to another element.

```
document.getElementById("overlay_1a").
  appendChild(createCircle(100, 200, 5));
```

Any graphical element, including grouping elements, can be attached under another G element or even the top SVG element itself, forming hierarchies. The order of rendering is strictly the order of appearance in the DOM, so the way each graphic part is placed relative to the others can be controlled. To visualize each management overlay, elements that belong to this overlay are grouped and the group is placed in the appropriate position inside the document.

From the point where an element has been created, many of its properties can be manipulated afterwards to alter its appearance. To change the color of a filled polygon, the application would access its style as an object property and modify the style property that corresponds to the internal fill color. Almost all conceivable style types are available in the SVG specification, such as fill color and pattern, line stroke and pattern, object opacity, etc. The easily accessible style model eases the programming of visually rich graphs immensely. For example, the code that would alter the opacity of an element would be like:

```
element.style.opacity=1.0;
//make object fully opaque
```

Underlying Operation.

The parts that modify the SVG graphical content are usually utilized as primitives by the parts of the application that handle all the events, including all user interaction. From the point in time where it is created, an SVG page produces events that trigger the execution of code

```
function createCircle(x, y, r) {
    //create the circle
    var c = document.createElementNS("http://www.w3.org/2000/svg", "circle");
    //set some attributes
    c.setAttribute("cx", x); //x coordinate
    c.setAttribute("cy", y); //y coordinate
    c.setAttribute("r", r); //radius
    c.setAttribute("style", "fill:#f34916");
    //create the grouping element
    var group = document.createElementNS("http://www.w3.org/2000/svg", "g");
    //attach element c
    group.appendChild(c);
    return group; // return the group
}
```

Example 2: Creating a circle.

that the programmer has placed as handler of these events. The event with the biggest influence is of course the `onLoad` event which basically triggers the creation of the entire page. As mentioned earlier, the document is initially devoid of actual graphics as this is completely handled by the application code. The tasks assigned to the `onLoad` event are summarized as follows:

1. Loading of the Configuration Document and interpretation of its contents. The configuration document will include a URI to a node topology description document. Additionally, several structured references to URIs of Management Overlay Documents will also be present.
2. Based on the URI of the node topology description document, the nodes are placed on the map using appropriate symbols and their positions will be stored in an internal array for later reference. All the graphical elements that represent nodes, including their labels, are placed in a separate overlay.
3. Each Management Overlay Document reference inside the Configuration Document includes information on how to handle the specific overlay. The administrator will typically want some overlays to be visible from the start, some to be available through a menu and some to be available through specific events, such as hovering or clicking on other parts of the map (nodes, arrows, etc.).

Node description documents and overlay documents contain simplified interaction information that controls the events that will trigger the loading of other overlays.

Overlay documents mostly contain information about arrows, labels and other shapes that their position revolves around the position of nodes. A rule of thumb is that the node description document contains elements that have an absolute position and are (almost) always visible, while overlay documents contain elements that have a relative position that is always calculated according to the position of other elements. The appropriate handlers will create all these artifacts and attach them on their respective overlays using the method that was illustrated earlier.

The application follows a strict strategy of loading overlays only when needed and never in advance. This modular way of operation ensures that the software will consume resources only when there is a clear need. Loading all the overlay documents and caching them in advance would be ill-advised, as in a complex scenario many hundreds of small size overlays may be present. Consider for example a multicast topology network map. The main overlay can be containing the underlying topology of the network (usually derived from the PIM [22] neighbor tables of each node), while there can be a secondary overlay depicting the distribution tree of a multicast group (or, even more, a source/group pair). In many cases there could be hundreds of group addresses present in the multicast routing tables of the network routers, which means that equally as many overlays would have to be available to the user.

Likewise, the reloading strategy of overlays must be carefully chosen. Considering the fact that each XML file that supplies an overlay is being produced on the server side periodically, it would be pointless to reload more frequently than the rate by which the data is refreshed. For that reason, the application will always follow the individual refresh rate of each overlay as it is supplied from the corresponding overlay document. This means that each XML overlay description document has builtin the refresh rate and the applet that produces it at the server side may choose to alter its refresh rate under various circumstances.

The last mechanism of loading overlays is through user interaction on the map itself. For example, hovering or clicking on a network link may activate an additional overlay that is downloaded at that moment and becomes visible. That overlay could incorporate additional information about the traffic that flows through the link and depict it with various means, like a conceptual traffic diagram over time or a report that manifests itself with a pop-up text box near the network link. It is left to the imagination of each implementor of applets, to devise new smart ways of revealing new information in response to user events.

The management overlay document schema allows the possibility to define pointers to the URLs of additional overlays which in turn could define other pointers and so on. A virtually unlimited depth of overlays that potentially activate other overlays can be created that way. Although this capability is not currently implemented, it is certain that very interesting ideas may be derived from the concept. The map could also incorporate overlays that could activate new areas that contain even additional nodes the ones defined in the node description document. An interesting example of this capability would be to have the initial map depict the layer-3 topology of a managed domain, and program the behavior where clicking over a link between two layer-3 nodes would reveal additional layer-2 devices that are positioned between them. For instance, clicking an arrow between two routers reveals the switches that are used to connect them.

Development Platform

The client SVG application is being developed on top of the Mozilla SVG implementation, usually inside the latest stable release of Firefox. The Mozilla implementation is ideal for the development of applications like these, because it already possesses an engine that can understand most of the SVG specification and, additionally, has excellent debugging capabilities built-in. Of these capabilities, the DOM inspector is surely the most important, as it allows to explore the SVG DOM in a tree-like fashion and experiment with changes or observe the effects caused by newly introduced application code.

Initial Management Applets

To test the code and provide real world proofs of concept, a series of management applets have been developed providing various categories of network information.

It is of paramount importance to explore various use cases of our presentation system and test different scenarios of operation regarding our applications.

1. A layer-2 topology discovery management applet has been tested on our university's campus to provide a physical view of the interconnections between all our managed switches. The topology discovery algorithm is based on CDP [23], and the node layout is based on the GraphViz package. This is a good example where an automated layout is most useful because of the large number of nodes that are depicted in the graph. An example can be seen in Figure 5.
2. A multicast topology discovery management applet has been developed based on the experience gained from the development of the multicast weathermap project [10]. The Protocol Independent Multicast neighbor information is used to find the PIM neighbors of each managed nodes and discover the multicast topology of the network. This module is under testing at the Greek Research Network. To give the opportunity to approximate the geographic positions of the routers, layout is handled by hand as their number is relatively small. An example of the visualization produced by this applet can be seen in Figure 4.
3. An IPv6 topology discovery management applet is also available. Using newly available IPv6 MIBs [24], this application can use the prefix

information and IPv6 neighbor discovery to lay-out an accurate depiction of the IPv6 layer-3 topology of a network. As in the case of the multicast applet, testing is been carried out on the Greek Research Network.

Strong Points

Security

From a general perspective, the client-server way of operation of this application coincides with the way of operation of many other software applications with similar requirements. The management data collection from the managed nodes is done solely by the NMS server, which of course increases the security of the system. Additionally, as explained before, the management applets produce XML management documents and place them inside the document directory of a simple HTTP server where they are available for retrieval. The fact that the application runs entirely in the client side and retrieves these static files which have been produced asynchronously, completely isolates the NMS server from the clients, resulting in a very secure scheme.

Scalability

It is expected that the scalability of the system in regard to the number of users that will be able to use it simultaneously, will be excellent. As explained previously, the management documents are simply placed inside an HTTP server. So, the number of clients that use the system

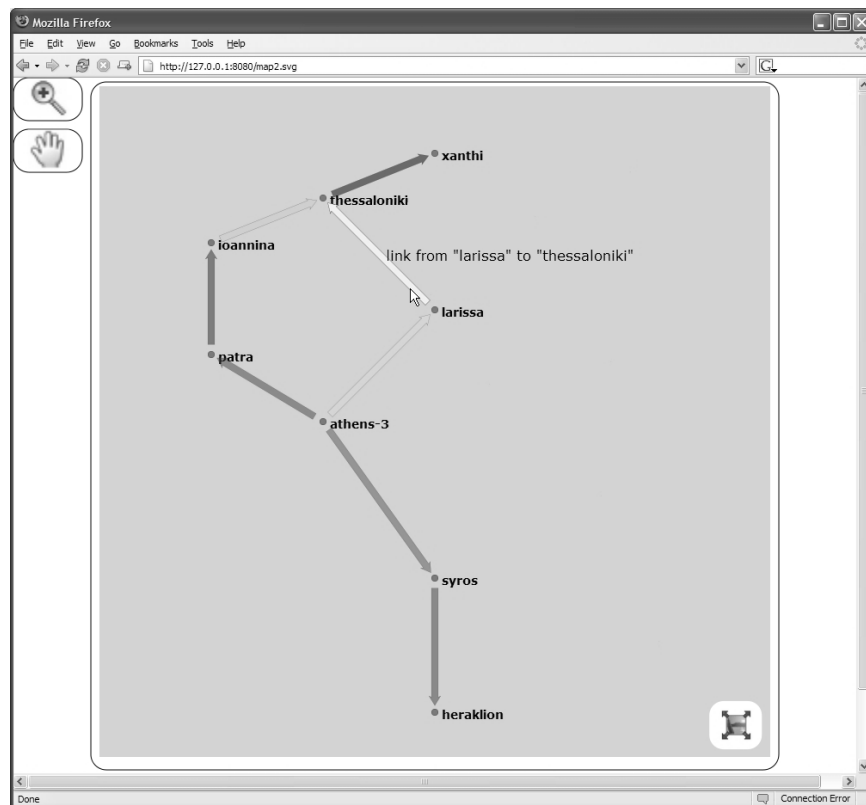


Figure 4: Prototype depicting a multicast topology overlay.

does not affect the operation of the management applets, because these two subsystems are independent from one another. In fact, the HTTP web server could very well be inside a different machine from the NMS server, so even large amounts of HTTP workload would not be placed on the latter. The fact that this architecture can handle a fairly large number of clients, makes it also appealing for creating interfaces that are publicly available. Security is also enhanced as the clients which generally reside on the Internet need not communicate using special protocols, but only simple HTTP operations. Additionally, there are no server side scripting technologies (like CGI scripts, JSP, PHP, etc.) involved, which makes operation of the web server very secure.

It is also expected the the user experience in terms of client speed and responsiveness will be enjoyable. The size of the application code is relatively small and, as mentioned earlier, the application will download new overlay information only when needed.

Related Work

GraphViz

The excellent GraphViz [21] package from Lucent incorporates some very sophisticated algorithms for laying out graphs. Although the tool is in no way associated with the creation of user interfaces or network management, its ability to export its calculated layouts into various formats, including SVG, makes it quite suitable for creating visually impressive network presentations.

As stated before, our tool does in fact use the GraphViz package for the calculations of layouts that contain many nodes. As the GraphViz package does not concern itself with interactivity, it is exceedingly difficult to make it create SVG depictions that incorporate even the simplest forms of user interaction.

Google Maps API

An application that is mostly strange to network management but exhibits notable conceptual similarities with our tool is the Google Maps API [25] which was recently released from Google. Using this API, the Google Maps product can be used as the basis for building other applications. The programmer can create visual artifacts such as arrows, polylines, and place-marks of various sizes and styles and place them on particular areas inside the geographical map.

In principle, this is completely analogous to our solution, where the vertices and edges are defined inside the node topology and overlay description documents. What is even more interesting is the fact that, in the Google Maps API, the user defined elements can be organized into different layers to produce more complex results. Again, this is similar to our own overlays. Combined with the fact that this information is described through XML files and is retrieved with AJAX techniques, the distinct similarities between the two tools can be easily observed.

As an aside, the Google Maps API does not rely on SVG or any other vector based drawing ability on any

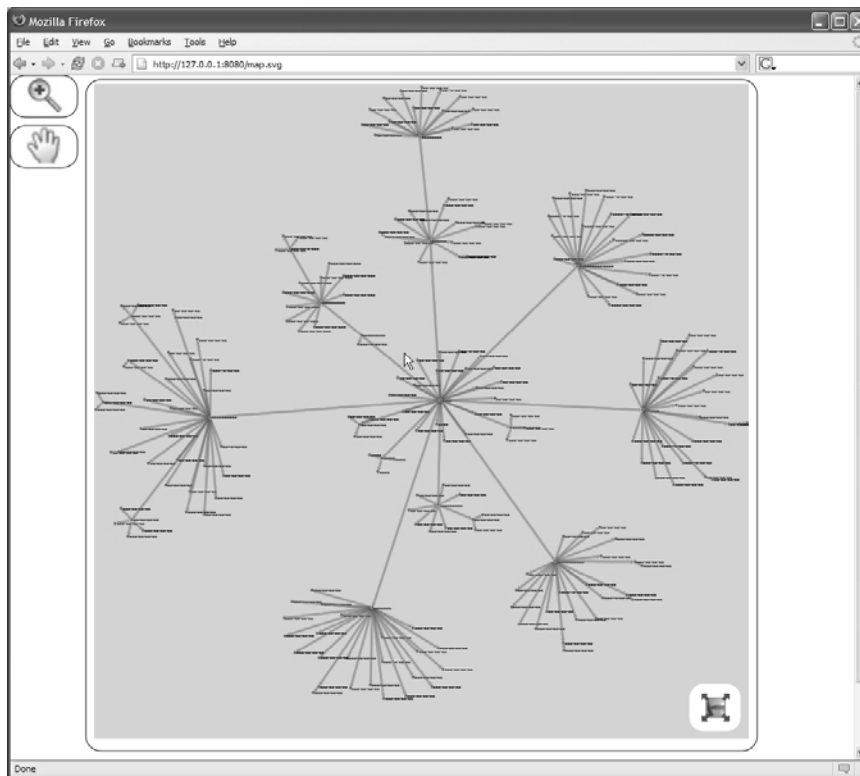


Figure 5: Prototype application depicting a large switched network. User can pan or zoom using the controls on the left.

browser (although it can utilize VML on IE) but uses clever techniques to supplement the lack of those abilities. We believe however that when the usage of SVG becomes more commonplace and is supported natively by the majority of web browsers, it could very well replace all these server aided techniques.

Carto::Net

Carto:Net, a popular web site that provides tutorials on SVG and its usage on cartography, has recently released a tutorial paper entitled "Dynamic Loading of Vector Geodata for SVG Mapping Applications Using Postgis, PHP and `getURL()/XMLHttpRequest()`" [26]. Inside the tutorial, there is the description of a geographical map application that dynamically loads vector data into the map each time the user zooms or pans inside the map area.

While the main focus of the application is of course related to cartography, the fact that data is fetched using AJAX methods and the SVG content is dynamically created on the client side and placed on the map, shows that the way of operation is again here very similar to our architecture.

HP OpenView

HP OpenView [27], one of the oldest and most popular network management framework available, employs the concept of an abstract visualization interface API to provide other network management tools with a platform which can be used to build native (X Windows) or Web based (through Java Applets) user interfaces that concern network management. This API, dubbed "OpenView Windows," allows the programmer to organize network management information into visual submaps that form a hierarchy. Each submap typically incorporates various nodes, unidirectional or bidirectional edges of various colors and styles and other similar constructs. As mentioned earlier, using a graph is the most common approach to depicting a network. A typical network management tool that plugs in to OpenView would have to initiate a series of calls to the API that would create the submaps, populate them with graphs and associate certain actions, such as clicking on a node, with opening other submaps or carrying out other tasks. The Network Node Manager product for instance, uses the OVW API to present its interface to the user. Of course, today there are many more network management tools that use the OVW API as a platform, contributing to its major success.

Certainly, our system has similarities with the OVW API in many regards. The provision of a visualization API to which other tools can incorporate their management information, the usage of graphs to depict networks and the association of user actions to various interactive features are traits that are certainly shared. However, there are still key points that differentiate the overall experience both from the view of the management tool programmer and the end user. For one thing, OVW is a regular API, which means that the programmer has to use a suitable language binding for it, while our system

takes a somewhat more liberal approach and imposes no restrictions on the internal architecture or implementation of the management applets. Using the XML management documents that were described earlier, does away with the need to have an API at the language level. On the other hand, the concept of submaps in OVW is superficially similar to our own overlays. It should be noted however, that a closer look reveals that each OVW based tool must create a different set of submaps and that each set can share no information with other groups. This in essence means that all the tools that plug in to an OpenView OVW installation not only work separately but also appear to be separate from the user perspective. In our approach, we sacrifice some of this compartmentalization and have all the overlays share the same node topology (which is described by the node topology description document) and draw on the same area.

Future Work

SOAP

Once the development of this system is finished, an interesting area to explore will be the inclusion of functionality that provides the ability of two way communication between the client applications and the server. Until now, the architecture clearly dictates that information flows only from the server to the clients. Adding the capability to let the client communicate information back to the server would enable possibilities such as saving various client side settings and retrieving them at the next session, executing various management functions on the management nodes, or even communicating operation instructions to specific server applets. The idea of using HTTP POST operations or even SOAP [28] calls seems like the right way to go on that regard. Modern browsers will most likely support SOAP operations (Mozilla already does in a limited fashion), so this is a very interesting possibility.

Management Document Hierarchy Simplification

The distinction between different types of management applet generated XML documents is also a point of extended discussion. It is quite possible that at least node descriptions and overlays may merge in a common schema which can be easily handled by a unified handler on the client side. The real problem is having different applets cooperate efficiently on the server side for the creation of these super-documents. It could be argued that merging those two schemata completely, could require further cooperation between the applets that produce them, thus making their development more cumbersome. This is definitely undesired, so great caution must be exercised in that area.

Currently the client application uses a very simplified model to handle various user interactions as they are declared inside the overlay description documents. As a generalization, a full meta-language that defines actions and events could be developed that could lead to substantially richer and more complex presentations.

The initial thoughts about this meta-language is that it will be surely mapped to the overlay document schema and that it will be similar in its feel to a functional programming language.

SVG Animation Capabilities

On the graphics plane, the usage of animation capabilities that are defined in the SVG specification may lead to an even more impressive presentation of network management data. Unfortunately, some SVG implementations do not have these capabilities yet, a fact that makes development troublesome.

Conclusion

Our experience indicates that the level of complexity and diversity to which the modern network management landscape has come to, often places a difficult task on the network administrators. In addition, these people often have to deal with complex management systems and, even worse, cumbersome and non-intuitive interfaces. The solution that is proposed on this paper tries to borrow ideas from the web by using the browser as a rich client and utilizing modern techniques like SVG and AJAX to provide attractive graphics. At the same time, the difficulty of extending the system is exceptionally low. Indeed, the network administrators can implement their management applet using any method or programming language they desire, and install it easily. Even better, the Javascript application which renders the graphics on the client side, will rarely require modifications. Combined with the fact that the XML schemata that govern the communication between the server and the clients were deliberately engineered with simplicity in mind, it is easy to reach the conclusion that the barrier to extend this system has been kept low. The application delivery method that is presented in this paper exhibits similarities with the classical approach of using Java applets to implement a rich client inside the browser. However, the protocol that is used to transport information is simple HTTP and the content is more presentation driven than management driven. Arguably, this strategy can lead to more compact communication volumes. The content is also encapsulated using XML, which does away with the need to have a special protocol and parser. Lastly, extensibility is benefited because of the usage of standards and openness of the architecture.

The demonstrated method of using AJAX to produce an interactive application that downloads its parts on demand, along with the usage of SVG to create vector based graphics that change during execution and respond to user actions, outlines a philosophy for the development of future web based applications. Furthermore, the method of overlay loading which has been outlined in this paper, advertises the strategy of loading graphics and interaction data on demand as the application is used. So, it is our hope that with our case we have inspired new ideas and innovative approaches on other problem areas besides network management.

Author Biographies

Athanasios Douitsis, born in 1976, graduated from the Department of Electrical and Computer Engineering of the National Technical University of Athens in 2000 and is currently a Ph.D. candidate at the Network Management and Optimal Design Laboratory at NTUA. He has been working for the NTUA Network Operations Center since 2000, involved in the administration of the NTUA campus network, the Greek Research Network (GRNET), the Greek School Network and the Greek Student Network. He has experience in Network Management, Monitoring and Measurements, Multicast, IPv6, VPNs and system administration.

Dr. Dimitrios Kalogeras was born in Athens in 1967. He graduated from the Department of Electrical and Computer Engineering of the National Technical University of Athens (NTUA) in 1991 and in 1996 he acquired the Doctoral diploma from the same department. Dr Kalogeras has participated in numerous research programs of the EC and the General Secretariat of Research and Technology in Greece. He has pioneered in the design and development of the NTUA and the GRNET data networks and is a member of the technical and scientific committee of the Greek School Network. Dr Kalogeras is a consultant on issues on networking and video signal processing. He is also the author and coauthor of publications in international magazines and proceedings of numerous conferences. From 2000 to 2002 he has served as a member of the Terena Technical Committee.

References

- [1] *Scalable Vector Graphics specification*, <http://www.w3.org/Graphics/SVG/>.
- [2] *Ajax: A New Approach to Web Applications*, <http://adaptivpath.com/publications/essays/archives/000385.php>.
- [3] *The X Window System*, <http://www.x.org/>.
- [4] *VNC*, <http://www.realvnc.com/>.
- [5] *Citrix ICA protocol*, <http://www.citrix.com>.
- [6] *Understanding the Remote Desktop Protocol (RDP)*, <http://support.microsoft.com/kb/186607>.
- [7] *SunRay Technology*, <http://sun.com/sunray>.
- [8] *Java Applets*, <http://java.sun.com/applets/>.
- [9] *GD graphics library*, <http://www.boutell.com/gd/>.
- [10] *The Multicast Weathermap*, <http://netmon.grnet.gr/multicast-map.shtml>, http://www.terena.nl/events/archive/tnc2004/programme/presentations/show.php?pres_id=47.
- [11] *Vector Markup Language*, <http://www.w3.org/TR/NOTE-VML.html>.
- [12] *Adobe Flash*, <http://www.adobe.com/support/documentation/en/flash/>.
- [13] *Adobe SVG*, <http://www.adobe.com/svg/>.
- [14] *Opera Web Browser SVG implementation*, <http://www.opera.com/products/desktop/svg/>.

- [15] *Mozilla SVG project*, <http://www.mozilla.org/projects/svg/>.
- [16] *Batik SVG toolkit*, <http://xmlgraphics.apache.org/batik/>.
- [17] *Java Web Start*, <http://java.sun.com/products/java/webstart/>.
- [18] *The Simple Network Management Protocol*, <http://www.ietf.org/rfc/rfc1157.txt>.
- [19] *Javascript*, <http://www.mozilla.org/js/>.
- [20] *The XMLHttpRequest object*, <http://xulplanet.com/references/objref/XMLHttpRequest.html>.
- [21] *Graphviz graph visualization package*, <http://www.graphviz.org/>.
- [22] *Protocol Independent Multicast*, <http://www.ietf.org/html.charters/pim-charter.html>.
- [23] *Cisco Discovery Protocol*, http://www.cisco.com/en/US/tech/tk648/tk362/tk100/tsd_technology_support_sub-protocol_home.html.
- [24] *IPv6 IETF charter*, <http://www.ietf.org/html.charters/ipv6-charter.html>.
- [25] *The Google Maps API*, <http://www.google.com/apis/maps/>.
- [26] *Dynamic Loading of Vector Geodata for SVG Mapping Applications Using Postgis, PHP and getURL()/XMLHttpRequest()*, http://www.carto.net/papers/svg/postgis_geturl_xmlhttprequest/.
- [27] *HP Network Node Manager*, <http://www.openview.hp.com/products/nnm/>.
- [28] *Simple Object Access Protocol*, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>.

Bridging the Host-Network Divide: Survey, Taxonomy, and Solution

Glenn A. Fink and Vyas Duggirala – Virginia Polytechnic Institute and State University

Ricardo Correa – University of Pennsylvania

Chris North – Virginia Polytechnic Institute and State University

ABSTRACT

This paper presents a new direction in security awareness tools for system administration—the Host-Network (HoNe) Visualizer. Our requirements for the HoNe Visualizer come from needs system administrators expressed in interviews, from reviewing the literature, and from conducting usability studies with prototypes. We present a tool taxonomy that serves as a framework for our literature review, and we use the taxonomy to show what is missing in the administrator’s arsenal. Then we unveil our tool and its supporting infrastructure that we believe will fill the empty niche.

We found that most security tools provide either an internal view of a host or an external view of traffic on a network. Our interviewees revealed how they must construct a mental end-to-end view from separate tools that individually give an incomplete view, expending valuable time and mental effort. Because of limitations designed into TCP/IP [RFC-791, RFC-793], no tool can effectively correlate host and network data into an end-to-end view without kernel modifications. Currently, no other visualization exists to support end-to-end analysis. But HoNe’s infrastructure overcomes TCP/IP’s limitations bridging the network and transport layers in the network stack and making end-to-end correlation possible.

The capstone is the HoNe Visualizer that amplifies the users’ cognitive power and reduces their mental workload by illustrating the correlated data graphically. Users said HoNe would be particularly good for discovering day-zero exploits. Our usability study revealed that users performed better on intrusion detection tasks using our visualization than with tools they were accustomed to using regardless of their experience level.

Defining the Problem

We believe that information visualization [Card, et al., 1999] technology can help system administrators wade through the tremendous amount of data they must review to ensure their systems are secure. At Virginia Tech alone, an estimated seven terabytes of packet data crosses our networks every day. If this data were printed out 66 lines and 80 columns to a page, double-sided, the daily stack would be 42 miles high. We interviewed 24 system administrators (selected by recommendation and happenstance) from two universities to determine if information visualization approaches could help them ensure the security of their systems and networks. Next we surveyed the literature to find out if existing tools fulfilled the requirements expressed by system administrators. Finally, we conducted usability evaluations on our own tools to make sure that we were offering what system administrators truly needed.

Interviewing System Administrators

We used a semi-structured interview protocol that covered the same general topics with each subject but did not always ask identical questions of everyone. Thus our results have given us insights that are probably accurate but are not easily quantifiable.

Four interviewees were our pilot group who helped us develop the interview protocol we used to interview the other twenty. Eleven of the interviewees also helped us with expert reviews of prototypes we had developed. We asked them for their biographical information, security-related duties, intrusion detection experience, tools they used in security monitoring, and what challenges they faced in keeping their organizations secure. A demographic summary of the interviewees appears as Table 1.

Interview Results

While the results of our prototype tests are presented in earlier papers [Ball, et al., 2004, Fink, et al., 2005], our interviews demonstrated that system administrators are not as homogeneous a group as we had initially thought. There were large differences in duties, tools, and techniques between those who primarily administrate servers and those whose main job is to assist users. There are also security officers, network analysts, and operations center specialists whose duties had much in common with other types of system administrators, but who had different areas of emphasis and different tool support needs. We would describe our “average” subject as a male with 14 years of experience (mean 13.39, stdev 5.25), charged with the care of ten or more servers with UNIX-like operating systems.

We learned that the single most important indicator of intrusions was aberrant communication patterns seen on the network or in host log files. The second largest indicator of problems was the appearance of suspicious processes on a host machine. We identified three kinds of analysis that subjects used to detect security problems, each at a different time relative to an incident:

1. Informative: When the administrator has no suspicion of malicious activity, she may use tools to periodically check the security state of her machines.
2. Investigative: When the administrator suspects malicious activity and is seeking to confirm it (e.g., after an IDS alert), she may use tools to gain a mental picture of the overall situation to focus further detailed search.
3. Forensic: When the user has confirmed the malicious activity and is seeking to locate the processes, files, etc., responsible for the behavior.

Interview subjects reported many types of analysis that we have grouped into four categories based on what data is being viewed:

1. Process Analysis: Looking for unusual names or numbers of processes on a suspect host
2. File System Analysis: Looking for unusual or modified files, especially executables, dynamically linked libraries, kernel modules, and service configuration files
3. Activity (log) Analysis: Using the log files to piece together what may have happened to a host in the recent past
4. Vulnerability Analysis: Using common or recent exploits and vulnerability reports as heuristics to find intrusions

Considering the times and types of analysis together, we can see when a particular analysis method is most and least applicable. Within each analysis time, we ranked the relative importance of each type of analysis to our interview subjects (Table 2). We decided to concentrate on Activity and Process analysis because they are the most applicable overall.

We were surprised that most of the interviewees preferred text-based tools to visualizations and other

high-end tools. This preference may reflect a population tendency toward low-level data analysis, but subsequent studies (especially [Fink, et al., 2005]) have demonstrated that our user community actually prefers visual solutions but has not yet found satisfactory ones. They typically did not trust tools that give an overview without showing the supporting details.

Analysis Time					
		Informative	Investigative	Forensic	Avg. Rank
Analysis Type	Process	3	1	3	2.33
	File System	4	2	1	2.67
	Activity (log)	2	3	2	2
	Vulnerability	1	4	4	3

Legend:

Most Important

Moderately Important

Least Important

Table 2: Relationship of times and types of diagnosis.

The administrators described how they identify suspicious communication patterns on the network (Activity analysis) and then manually trace those patterns back to a set of suspect hosts. On these hosts they would look at the processes running for suspicious activity (Process analysis). The users were determining whether communications were malicious based on what processes were initiating them. We realized that our users could save considerable time and mental effort if they could automatically trace packets seen on the network to processes running on their monitored hosts. While there are many tools in the literature and practice that claim to correlate host and network events, we were quite surprised to find that none actually did this packet-process correlation that was so fundamental to the work our users reported doing.

Key Conclusion: Packet-process correlation directly supports work typically done by our users, but no tools exist to automate this activity.

A Taxonomy of Available Tools

We reviewed the literature and current practice and organized the known tools by the context of the data they use and the way they present it. Figure 1 illustrates the resulting taxonomy that we use to compare tools. The fraction in the lower right corner of

Area (# Sub)	Job Description	Scope of Responsibilities
Servers (9)	Manages web, file, mail, compute, etc. servers with little or no user contact.	Average of 23 servers and 3 users.
Users (7)	Manages end-user workstations for individuals or labs; lots of user contact.	Average of 4 servers and 75 users.
Security (2)	Receives data from SAU/SAS, sets security policy, coordinates response to security incidents, forensic investigation, risk analysis, law enforcement liaison.	Indirectly responsible for entire university estimated at 300 servers and 40,000 users.
Network (2)	Monitors health and welfare of enterprise network; investigates ways to improve performance.	Responsible for network infrastructure, usage, and planning.

Table 1: Demographic summary.

each category in the figure shows the number of interviewed system administrators who mentioned using a tool from that category.

The abscissa (x-axis) of the taxonomy diagram has four views of communication context that security awareness tools typically employ:

1. The internal host view (IH) that presents data internal to monitored host(s) without regard to network connections.
2. The networked host view (NH) that displays data that concerns only the monitored machines, but includes the broader context of their network connections.
3. The network view (NV) that presents traffic data in the context of a network or internetwork.
4. The end-to-end view (EE) presents entire communications by interpreting process communication data on a networked host in the larger context of the network or internetwork where it resides.

Each of these communication context views is important, and none can substitute for the others. However, the networked host and the end-to-end views are seriously under-represented in the literature. The ordinate (y-axis) of the taxonomy diagram contains four ways security awareness tools may display information:

1. Text-based displays (TB) may have graphical user interfaces, but the information presented is pure text.

2. Dashboard-style displays (DA) present mostly text data, but use simple preattentive features like color and motion (blinking) to draw the user's attention to critical items.
3. Summary charts and graphs (CG) present abstract quantities like throughput pictorially via statistical graphs, etc.
4. Visualizations (VZ) convey the state of some object (a machine, a service, or an alert, for example) via an abstract marker or icon.

Although there are many other ways one could classify security awareness tools, this taxonomy clearly shows what is missing from the system administrator's tool chest. Our tool (the HoNe Visualizer) is the only known visualization of the end-to-end view and the only networked host visualization that is suited to security requirements.

The Need for Visualization Across the Host/Network Boundary

Our discussions with system administrators regarding security incidents and common problems gave rise to several usage scenarios where visual packet-process correlation would be beneficial. For example:

- *Detecting Covert Communications:* When we see network traffic for TCP port 80, we often assume that it is web-related, but with visual packet-process correlation, we could see whether the clients and servers are really web browsers and web servers.

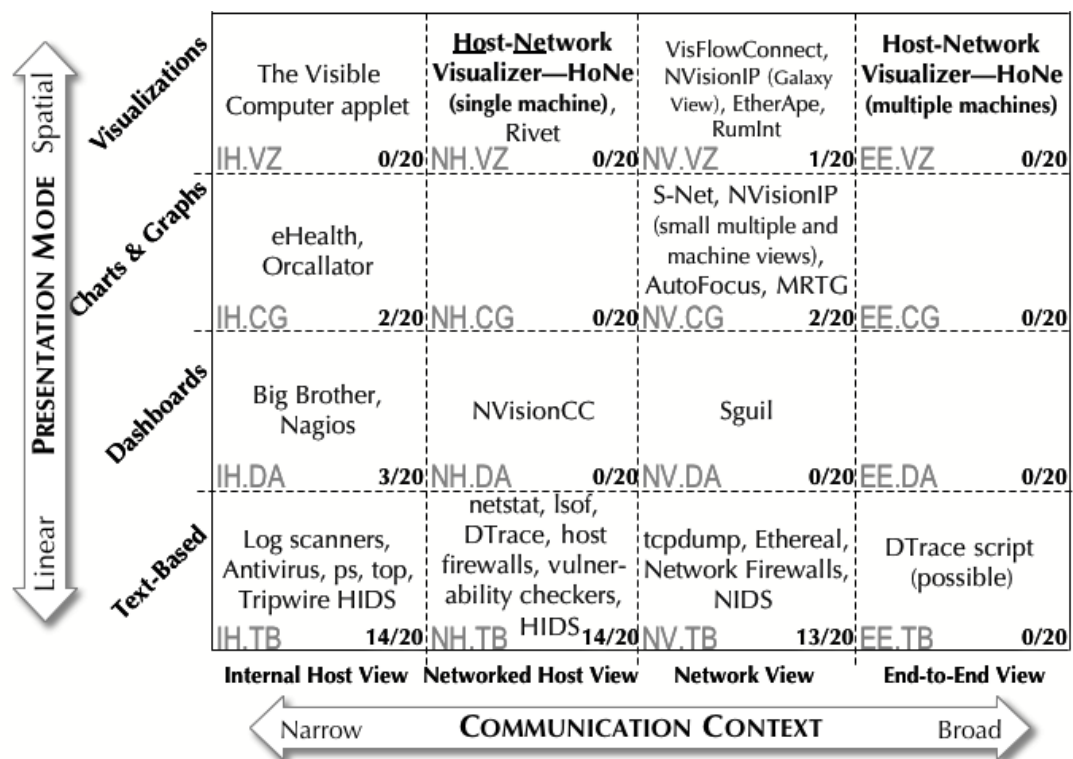


Figure 1: Taxonomy of security awareness tools.

- *Spyware and Adware*: Rather than running tedious system scans to locate spyware, a user could have ambient awareness of his machine's network activities and communicating processes.
- *Fine Tuning Firewalls*: Testing for vulnerabilities becomes more complex when one or more firewalls are between the tester and the target host. It would be helpful to be able to observe the effects of traffic on the target host via visual packet-process correlation.
- *Cluster Computing*: Administrators who maintain large cluster computers could use visual packet-process correlation to see communicating processes in the cluster and monitor for malicious activity.

In each case, visualization of packet-process correlation would benefit the administrator and user by complementing existing tools and enabling quicker diagnosis of problems. Even automated intrusion prevention systems could make better decisions if they could take the process names into account.

But Where are the Tools?

We surveyed tools of all kinds, free and commercial, and found our approach was unique. Here, we have only included a brief discussion of a few tools that are closest in apparent function.

Programs like Zone Lab's firewall, ZoneAlarm [ZoneAlarm], can tell a user which process a packet is emanating from on Windows machines, but it does not enable the same visibility from the network side. ZoneAlarm is more powerful than netstat because it enables the user to control connections. However, ZoneAlarm provides no visualization, nor can it provide remote monitoring of another machine. Foundstone's tools Fport and Vision [Foundstone] map open network ports to the applications, services, and drivers that opened them, but they cannot trace packets across the network, nor can they show this information about the open ports from the network side of the host/network divide. In contrast, tcpdump and other programs based on network sniffers, can catch and record every packet that a machine sees on the network but cannot tell the user whether the packets were seen or used by processes running on the receiving machine.

The lsof utility (<http://freshmeat.net/projects/lsof/>), created by Vic Abell of Purdue University, lists all open files related to processes running on UNIX machines. Because communication mechanisms such as pipes and sockets are considered files on UNIX, lsof can provide a very effective, if difficult to understand, view of communications on the local host. For instance, typing "lsof -i -U" lists all the processes with open Internet sockets and UNIX pipes. Lsof can report the process IDs, the file control block information, command names, and many other pieces of information that an expert can piece together into a very complete view of host communication activities. One important limitation of lsof is that it works by polling

rather than by continuously watching the file system. Thus, lsof may not show connections that are created and destroyed between polling intervals. Although polling intervals can be shortened to a single second, data collection takes a relatively long time and may block at certain system calls.

The netstat utility first appeared in BSD UNIX version 4.2 and has subsequently been added to DOS, Windows, and other operating systems. The netstat command textually displays the contents of various network-related data structures. There are a number of output formats of the command, including: a list of active sockets for each protocol, protocol traffic statistics, remote endpoints, and routing information. While lsof can display what files are open due to network activity, netstat can show the state of TCP and UDP sockets. On some UNIX-like operating systems and Windows, netstat can also tell what process the socket belongs to. Adding the process makes netstat comparable to Sysinternals' TCPview Pro [Sysinternals2]. Both lsof and netstat show communications from the host's point of view and both use polling. Thus, they may miss very short connections or communications that do not use sockets such as ICMP. Sysinternals' ProcessExplorer [Sysinternals1] is a Windows equivalent of a netstat/lsof amalgam, but does no more than either of these can.

The eHealth suite of network management tools from Computer Associates [eHealth] provides a multiple internal host view of medium and large networks of hosts using SNMP traps and queries. The product uses changes in its source data coupled with architectural configuration details of the network (provided by the user) to perform "root cause analysis" when traffic congestion problems occur. This analysis provides an overview of the management state of the system. The eHealth tools do not monitor traffic flows and relate them to host activities; rather they analyze host performance data and then use artificial intelligence to infer causes of potential network outages. Thus EHealth tools are essentially loosely correlated internal host views rather than networked host views or end-to-end views.

A staggering amount of effort is required to monitor computer activities for security. Most administrators we have interviewed care only about a few hundred machines they are personally responsible to maintain. But even with relatively few machines, having to examine anomalous events from one side of the host/network divide at a time can be time-consuming and error prone. No other known tool (freeware, academic, or commercial) at the time of this writing correlates network packets to the machine processes that generate it and visualizes the result. Some may integrate host information from multiple hosts on a network, and others may present network activity side-by-side with selected data from host logs, but none actually correlates each packet to a process on the machine that sent or received it and provide a visual presentation of this correlation. Thus, our approach is unique.

Solution

There are many approaches to fill in the gap left by existing tools. In this section we present the alternatives we considered and the final solution we arrived at to correlate packets to processes and visualize the result.

Alternative Approaches

On investigating why no packet-process correlating software existed in the literature or practice we discovered that the separation between host and network was enshrined in the networking stack inside the kernel. The problem is illustrated in Figure 2. Given a modern operating system kernel and network stack based on a layered networking model [OSIArch, RFC-1122] (Linux, Solaris, *BSD, Windows, and MacOSX to name a few), we cannot know both the source machine and the associated processes for a given incoming packet at the same time and place.

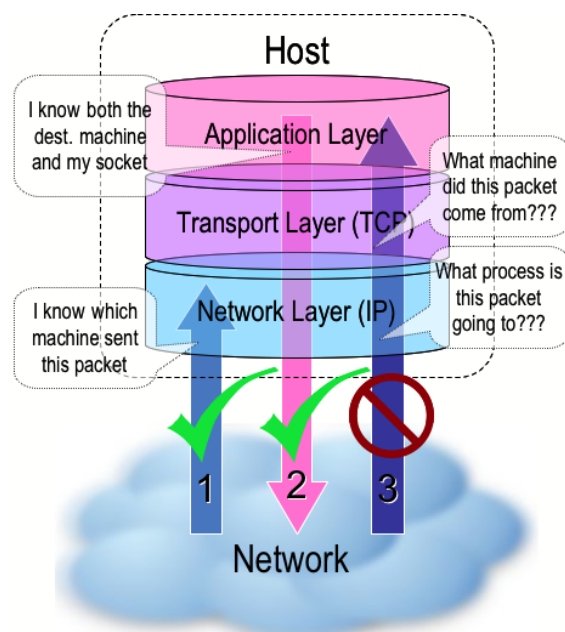


Figure 2: In network stacks based on the ISO layered networking model, we cannot simultaneously know both the source host and destination process for incoming packets.

This is not an oversight in the kernel design so much as it is an intentional separation of concerns characteristic of the layered approach [RFC-1122]. Modern networking models divide the work of a communication protocol into separate layers where each layer has the property that it only uses the functions of the layer below, and only exports functionality to the layer above. The set of layers is called the protocol stack. The logical separation makes the protocol easier to understand and allows separate vendors to provide different layers because they must agree only about the layer interfaces.

However, the layered model implies that the transport layer (TCP/UDP) may only call functions of the network layer (IP), and IP may only provide functions to TCP/UDP. To correlate packets to processes, TCP would need to call a function in IP that returned the source and destination address, but no such function exists in the standard because routing information is irrelevant to the transport layer. Another possibility would be for IP to call a function from TCP to find out what process the packet is destined for, but this directly violates the layering principle. Thus the necessary correlation could not be obtained in a faithful rendering of the standard protocol stacks.

For incoming packets, (number (1) in Figure 2) we know the source host at the IP layer only. When the packet reaches the transport layer (as in arrow (3) in Figure 2), this information is no longer available. In the application layer there is no problem for outgoing packets (number (2) in Figure 2) since the destination host is known, but once a packet reaches the network layer, its source socket (and thus process) is no longer known. So the correlation engine could not be confined to any single layer in the protocol stack. This inconvenient fact implied modifying the kernel itself, something we were loath to do. The following are the implementation alternatives we tried:

- Integrating process data from tools like netstat and lsof with packet data from tcpdump into a single picture
- Modifying Glibc to intercept all socket() calls and tracking the processes that made the calls
- Using firewall redirection rules and divert sockets to copy all network data to an accounting process
- Modifying the kernel itself

Integrating netstat And lsof With tcpdump

The first solution is very complete except that netstat and lsof rely on timed polling and can only provide snapshots of the machine's state. Thus, extremely rapid events like infection by the SQL-Slammer worm can easily slip by unnoticed. By the time tcpdump had captured a packet, the socket that created or received it would be gone and with it any hope of connecting the packet to a process. Based on our attempts to detect the nmap port scanner running on the monitored machine via netstat and lsof, it appears these tools cannot track incomplete or very brief connections. Thus, most scanning behavior (inbound as well as outbound) would be missed. Outgoing packets that were rejected by the firewall also never created an entry in netstat's list of sockets, implying that attempted communications originating from some unauthorized process on the monitored host (a very important indication of intrusion) would be missed. Finally, this approach is unsatisfactory because the host and network data must be retrieved separately and thus will have separate timestamps. Any packets that cannot be matched to a running process are suspicious, and if the data collection is not

truly integrated, mismatches due to the separate data collection will produce false positives.

To see whether or not netstat could capture all the information needed, we wrote a script that retrieved random web pages at normally distributed random intervals ($\mu = 5$ sec, $\sigma = 1.5$ sec). We collected and examined the netstat data at the maximum frequency while the script was running and found that whole connections were missed. We then modified the netstat source code removing the sleep interval to see if its coverage improved. The resulting program dominated the processor and generated 22 MB of text per minute when we ran it against only the web surfing script. The output had very little entropy, evident from the fact that it could be compressed by greater than 99%. Even with this extremely fast collection rate, netstat still missed parts of the web conversations and important socket state transitions. Thus we determined that netstat was insufficiently accurate even for low data rates no matter how frequently it collected data.

In contrast, tcpdump catches all packets received by the machine (unless the kernel drops them because of overload). So it would be theoretically possible to use tcpdump packet traces to recreate conversations that netstat was missing. However, consider that an attacker could use this knowledge to plant fake (spoofed) conversations on the network to make it look like a connection was in a different state than it truly was. Then when a packet appeared that did not match any connection seen by netstat or lsof, the tool would have to choose between believing tcpdump (subject to spoofing) or netstat (subject to missing events). The netstat/lsof/tcpdump solution is incomplete and too loosely integrated to be useful.

Modifying Socket Libraries

Another approach would be to modify the program libraries responsible for creating and maintaining sockets (particularly Glibc) to maintain a list of all open sockets and the programs responsible for them. This approach would work for all executables that were statically bound to the socket libraries, but it would not be able to track the activities of processes whose executables were dynamically bound, nor would it prevent programs from avoiding our modified library by using one of their own. In an earlier usability study we found that bypassing these libraries was a common tactic used by malware authors to avoid detection.

Firewall Redirection

To ensure that we accounted for all the packets while the sockets were still active, we needed to move closer to the kernel. By adding packet redirection rules into the monitored host's firewall, we could copy all traffic to a special "divert socket" owned by an accounting process and correlate each packet to a running process or determine that no such process exists.

A shortcoming of this approach is that using current firewalls with packet-diversion capability, (e.g., NetBSD's ipfw and Linux's iptables) one must reinsert

the diverted packet back onto the machine's networking stack for processing. Reinserting the packet at the proper place in the firewall's rule set without changing the effective behavior of the firewall is very tricky. This method also contacts the accounting process for every packet that enters a monitored interface regardless of whether the machine is listening on that port. Thus, the accounting process becomes a performance bottleneck and makes the monitored machine much more vulnerable to denial of service. Any instability in this process could tie up the firewall and crash the machine making the firewall-divert-socket approach inherently unstable.

Kernel Modifications

Because none of the other methods worked, we were constrained to modifying the kernel itself to make the required packet-process correlation. The modified kernel intercepts and tracks each packet that belongs to a running process. This approach has the advantage of showing only the data that actually had an effect on the monitored host. "Script kiddie" attacks and other noise that doesn't affect a process on the machine simply don't appear. We chose an implementation that had the smallest performance impact on the kernel. We simply log the header of every packet associated with a socket and the name of the process that created the socket to a text file. We then process the text file into an SQL database and visualize the data via a Tcl/Tk user interface. The performance impact to the machine is comparable to running tcpdump, and there is no impact when the kernel module is not loaded.

An alternative to the modified kernel that we did not try would be an instrumented kernel, where certain actions such as receiving a packet triggered handler routines. These handler routines could be used to do the packet-process correlation. A script using Sun's DTrace [Cantrill, 2004] may be capable of such kernel instrumentation without modifying the kernel at all. At the time we created the bridging architecture, we were unaware of DTrace, but using it instead of custom kernel modifications could be an excellent direction for future research.

HoNe's Architecture

We used Linux as our development platform because it is open-source and its kernel is relatively well documented. We planned for HoNe to be able to remotely visualize data from other machines by separating the user interface, data collection engines, and database functions into distributable components. Figure 3 shows a block diagram of the overall architecture.

The HoNe Visualizer is the key control component of HoNe. The user can use it to load and unload the loadable kernel module and to run the database builder to update the connection database from the connection log file or external data sources.

When the kernel module is loaded, it registers handlers for Netfilter hooks for all incoming and outgoing packets. Each incoming packet triggers the

Local_In handler. If the packet is a TCP or UDP (IPv4 only) packet the handler copies the IP header plus the first 20 bytes of the TCP or UDP packet to a log buffer, records a high granularity timestamp (via `gettimeofday()`), retrieves the process identifier (PID) of the socket owner, and queues the message for logging to the connection log file. The module performs similar work when an outgoing packet triggers the Local_Out handler.

Part of the difficulty with this approach is that Netfilter exists in the IP layer of the communication stack, while the information about the process that owns this socket is in the TCP layer. Thus, for incoming packets, we had to expose a private TCP function in the kernel `dv_ip_v4_lookup()` which retrieves a pointer to the TCP socket data structure. This pointer is readily available for outgoing packets. We added to the socket data structure a reference to the socket's owning process (actually its creator) that we fill in when the socket is created. We do the same for UDP, but we do not handle other protocols.

Whenever the kernel module is called, it checks to see whether there are any log records to write. If there are, it formats these and writes them to the log file. As part of the formatting process, the kernel module converts the socket owner's process identifier (PID) to the filename of the process's executable. This makes it possible to the user to locate executable files that are causing problems. The log records contain a timestamp, the PID and name of the owner process, the socket state, the packet size, and the packet header.

We convert the packet header to a hexadecimal text string so the log file can be a text file. We could also log to an offsite analysis server via a socket, but we have not yet implemented this. When the kernel module is unloaded, it records the number of packets it processed and how many it dropped.

Description of a Successful Hack

In this section, we introduce HoNe by showing what happened on a monitored machine as it was hacked. This was a real incident, not a simulation. Figure 4 shows the state of the machine during the critical first 20 minutes of the intrusion.

In Figure 4, the user has opened a connection database for monitored host 128.173.54.204 and focused on the area of interest using the Day, Hour, and 5-minute overview histograms on the right. On the left is the detailed view of what happened in the critical first minutes of the hack. The events are roughly in time order from top to bottom, so we can trace the progression. First the intruder logged in from remote host 81.196.144.243 using a password he had broken earlier. We know he already had the password because there is only one login attempt. Next he started `ftp` to download files (probably his toolkit) from a remote machine. Finally, he started up his exploit program (evidently an Internet Relay Chat (IRC) 'bot) to allow other users to make connections to this machine. Using the IRC bot, the intruder later attempted to gain root control of the machine and use it to attack other machines. However, at that point the machine crashed and we stopped the experiment. Because we can see

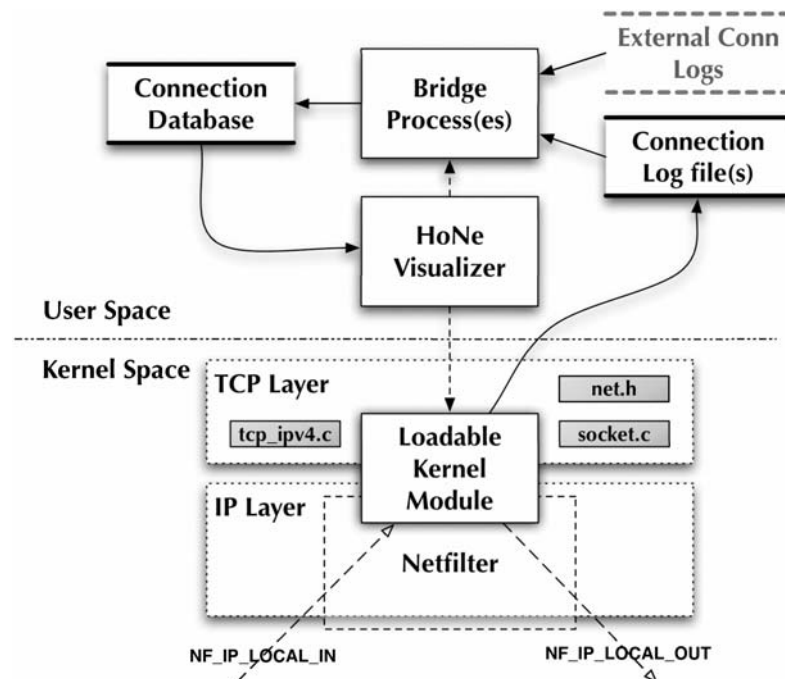


Figure 3: A block diagram of the HoNe architecture. The shaded blocks in kernel space indicate changes to the kernel itself beyond the loadable kernel module. Solid arrows indicate data flows and dashed arrows indicate control flows.

how the communication flows connect to processes, we get a clear picture of what is happening as a machine is being hacked. We see that the machine is talking to a remote server on port 8000 and that a non-standard “sshd:” process is running. But more than this, we see how these facts relate: that the fake sshd process is the one responsible for this communication. Thus, we can classify the communication as malicious with a greater degree of certainty.

The HoNe Visualizer

This section presents the HoNe Visualizer’s user interface and explains its operation. The main window layout shown in Figure 5 is divided into two halves: the left half is the detailed view (numbers 1 to 4), and the right is the control pane (numbers 5-8). The detailed view is subdivided into four trust regions: 1) Enterprise Managed, 2) Enterprise Unmanaged, 3) Foreign Managed, and 4) Foreign Unmanaged. Machine icons appear in the upper (Enterprise) regions when they belong to the user’s company. In the lower regions are the Foreign hosts. Machines that are administered and monitored by the user are considered Managed and appear on the left side. A machine should be considered managed if it is running the kernel module and is providing reports to the user. Machines on the right are considered Unmanaged, and appear in the lower regions. In the figure, there is only

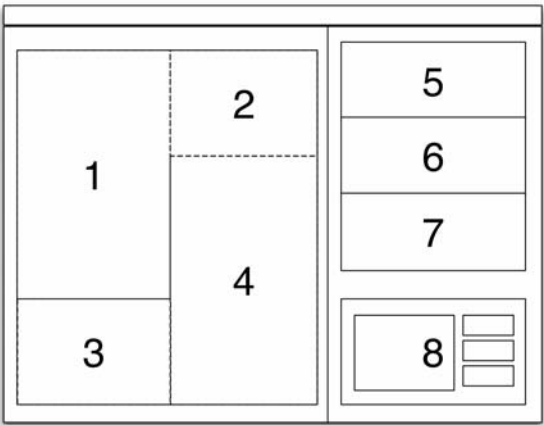


Figure 5: General layout of the HoNe main window.

one enterprise-managed host, 128.173.54.204. The other hosts are foreign-unmanaged machines. The user is responsible for entering the IP addresses and ranges of machines used for this classification into the configuration files prior to running the visualization.

The right-hand pane shown in Figure 5 contains three histogram windows at different time scales, 5) a daily view, 6) an hourly view, and 7) a five minute view. These overviews show the traffic levels and connections using a histogram for which each bar represents a day,

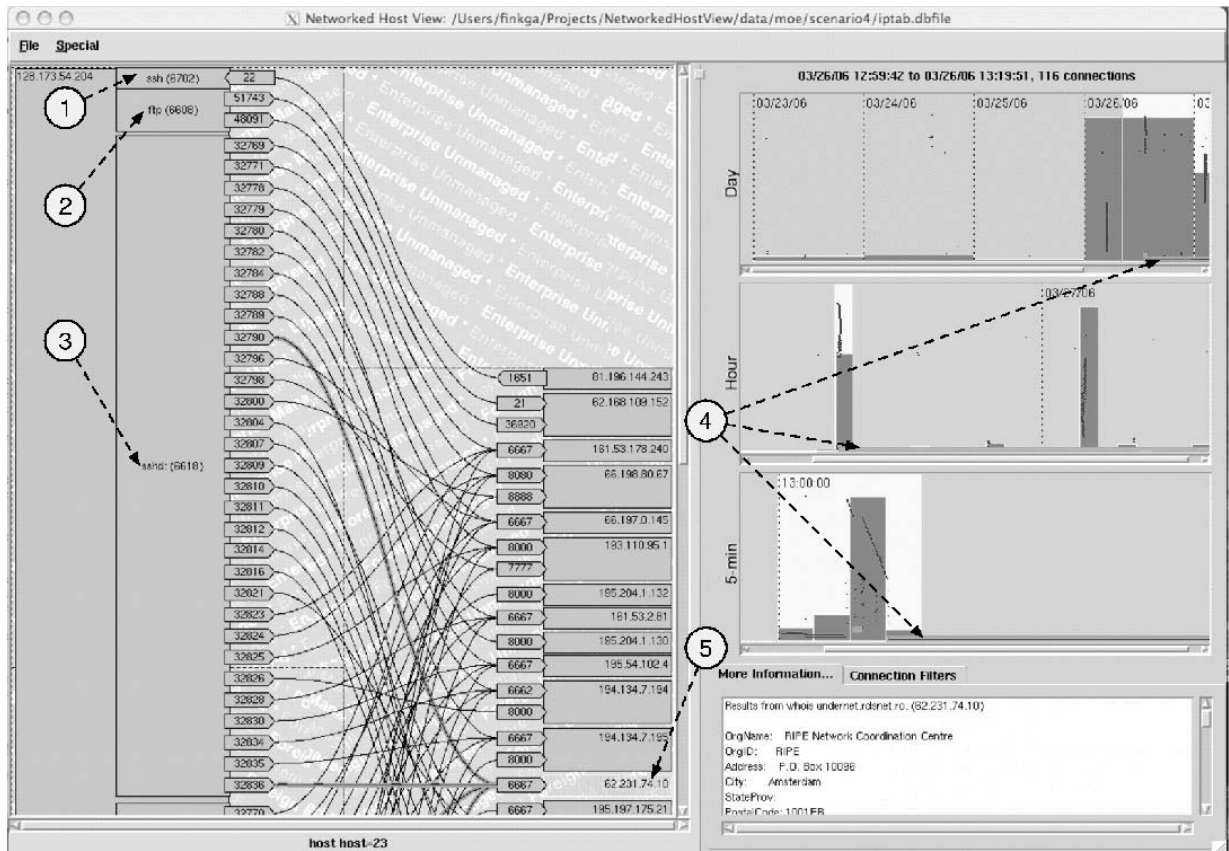


Figure 4: Snapshot of a successful hack: (1) Login, (2) downloading a toolkit, (3) starting the IRC ’bot, (4) suspicious, 23-hour connection, (5) Romanian IRC server contacted.

hour, or five-minute period of time. Item 8 is a tabbed pane with the more information window in one tab and a list of SQL filters in the other. All these items will be explained in later paragraphs.

In the following description of the visualizer's user interface, we will refer to another intrusion incident we captured using HoNe. Figure 6 shows an internal system (128.173.54.204, the virtual machine) as it is being hacked by several cooperating external machines. The status line at the top on the right side indicates the events shown occurred on 12 Jan 2006 from 02:43:50 to 02:57:30 in the morning. Note: These events appear in the detailed view slightly out of time sequence looking top to bottom. This is because a connection to one machine on the left occurs in the middle of several connections with another machine. We have elected to keep the machine icons together rather than to keep the time sequence intact.

First we see a barrage of SSH login attempts (highlighted points on the time window panes on the right) from foreign host 200.32.73.15. This barrage continued for 23 minutes. There had been similar barrages during the previous evening and four days prior. Because this SSH barrage continues after the actual hack has started and from a different IP address than the attacker who gets in, we assume it is an attempt at

covering the actual intrusion. The attacker has already guessed the password, probably from an earlier SSH login attack. The intrusion happens when the attacker logs in from 82.77.200.63. Next we see the attacker using wget to download a toolkit from 66.218.77.68, and a few seconds later, the machine is hacked, having opened up an IRC server (unnamed process 2232) and client (innocuously named "ntpd" but making foreign connections to port 6667. Because the entire intrusion took place within three minutes, including downloading and starting the IRC bot, we believe this attack was automated. Full forensic analysis was not performed for this incident, so this is not a definitive analysis.

The detailed display (enlarged in Figure 7) shows each host by IP address (and DNS name if known). In Figure 7, the host icons are items (1) and (6). Item (1) is the monitored host and it is in the enterprise-managed zone. The host icons pointed to by item (6) are foreign-unmanaged machines. For managed hosts, HoNe shows the processes involved in the communications displayed inside the host box (icons pointed to by item (2) in Figure 7). Processes contain the executable name and PID if known.

Bristling from the inside edges of the processes or machines are the ports the machines are using to communicate (pointed to by items (3) and (5) in

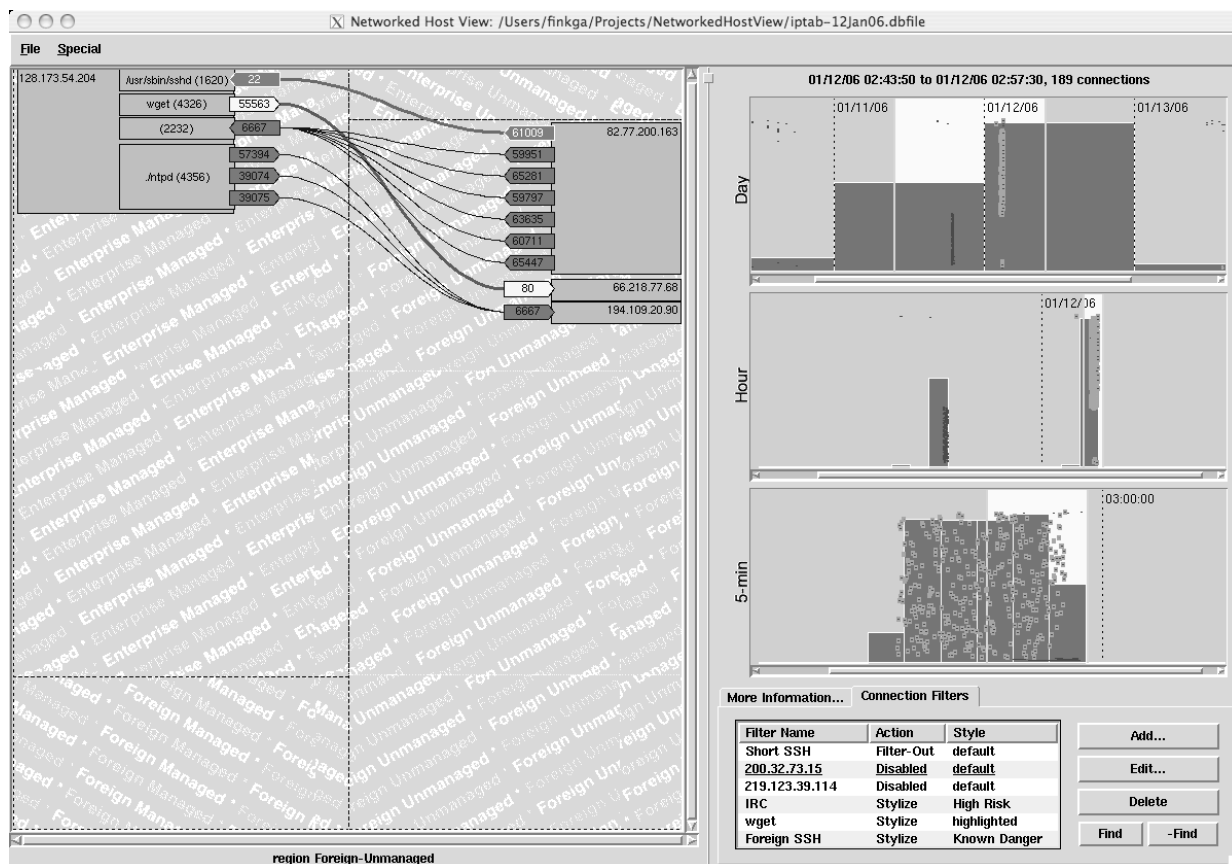


Figure 6: A screenshot of HoNe's visualization. The visualization displays connection data from an SQL database generated by the modified Linux kernel to visually correlate each packet to a running process.

Figure 7). Ports are shaped like arrows with listening (server) ports pointing toward their machine and initiating (client) ports pointing away. Item (3) in Figure 7 is a server port attached to the SSH service on the monitored host. Item (5) is a client port used by some unknown process on the foreign-unmanaged host, 82.77.200.163. The arrow shapes help users to clearly see who initiated each communication.

Communication lines (as pointed to by item (4) of Figure 7) join the client and server ports. Item (4) in Figure 7 points to a connection between an ephemeral port on the client side and the managed host's SSH server port on the server side of the connection. The arrow-shaped port icons point away from the client and toward the server. Communication lines and icons for ports, processes, and hosts may be colored manually or by some user-defined filtering expression. In this case, the user has elected to highlight any successful incoming SSH sessions initiated on the monitored machine by a host in the foreign-unmanaged zone with the "Known Danger" color scheme (white text on a bright red background).

HoNe provides multiple levels of drill-down so the user can keep the detailed view uncluttered. However, the nature of Internet activity makes it possible for thousands of activities to happen within a fraction of a second. Thus we enable users to zoom or shrink the icons in each region independently to fit them into a single screen.

The right pane of the main window primarily contains controls that determine what the detailed view shows. The top section shows three histogram

timelines that form the connection overview (see the close-up in Figure 8). The bottom section is a tabbed area where the Connection Filters and "More Information" panes reside.

The connection overview (Figure 8) shows the entire database file at a glance on three separate scales. The reason for separate scales is that users stated that they needed to distinguish events at the microsecond level, but when asked how much data they might look at to investigate intrusions, they said they might want two or more months' worth. These viewing levels cover about 13 orders of magnitude (from 10^{-6} to 10^7 sec). The tri-level overview plus the detailed view is how we chose to span this large magnitude range.

The overview was inspired by histogram sliders [Li and North, 2003], a new widget designed to give instant feedback to help the user locate where on the slider scale the most relevant information lies. The timelines represent the passage of time from left to right with earlier events placed closer to the left. The relative number of connections within the time period the bar represents determines the height of each histogram bar. We multiply height in pixels of the timeline by the number of connections within the time period of the bar divided by the total number of connections displayed in the whole histogram to derive line height of each bar. So the top histogram in Figure 8 has one bar that contains most of the connections for the whole four day period. Placing the mouse over a histogram bar shows a "tool tip" window with start time of the bar and the number of connections within its duration.

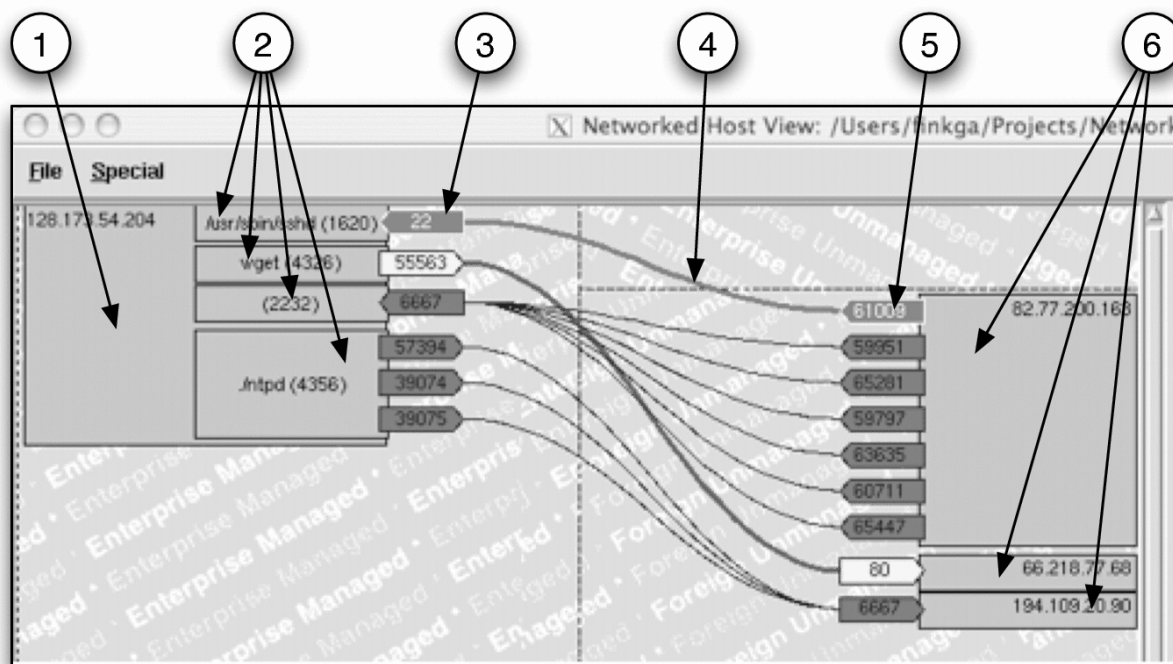


Figure 7: An enlargement of the detailed view of the HoNe visualizer.

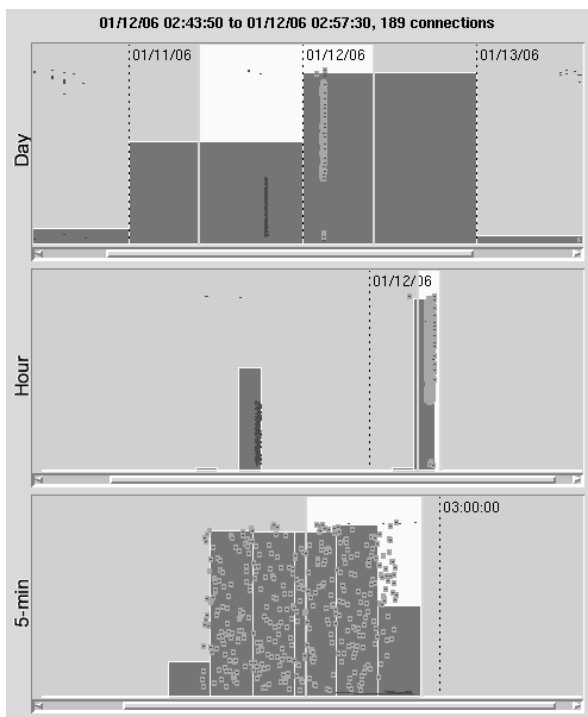


Figure 8: Detail of the connection overview histograms. Connection lines are brown but highlighted in cyan when selected. Selecting any connection highlights it in all views.

The pale yellow area bounded on the left and right by bright yellow bars is the area of interest for each timeline. Users can move the whole area of interest or slide just the left or right bars. The width of the area of interest is a focal area that determines the amount of time that will be displayed in the next lower view.

The topmost timeline's histogram bars each represent one day's worth of data. Fifty or more days' worth of data can be shown here concisely. When the user selects an area of interest in this histogram (of no more than 1.5 days in duration), HoNe displays that area in the second histogram, where each bar stands for an hour of data. Similarly, selecting an area of interest (no longer than six hours in duration) here brings up the bottom histogram with a bar for each five minutes of data. When the user selects an area of interest in this lowest histogram, the detailed view pane shows the hosts, processes, ports, and communication lines in that time window. The information line above the histograms tells the extent of the finest-granularity area of interest that the user has selected and the number of connections it contains.

The histograms are overlaid with horizontal dark brown lines that represent individual connections. The horizontal length of the connection line represents the duration of the connection. Because the scale may force many connection lines to zero length, we have constrained all lines to at least two pixels length. The longer duration a connection has, the nearer it is placed

to the bottom of the histogram. The metaphor we use is "longer connections are heavier and they sink to the bottom." Placing the mouse over a connection line shows a "tool tip" with the source and destination IP addresses and ports and the measured connection duration in seconds.

When a user selects a connection or executes a search, the selected or matching connections are highlighted with a cyan border. In Figure 8, the user has executed a search to highlight all SSH connections initiated by foreign-unmanaged hosts to any managed host. The matching connections are highlighted in every view (detailed and all three histograms) where they appear.

The bottom section of the controls pane has tabbed windows with "More Information" (Figure 9) and Connection Filters (Figure 10) tabs. The "More Information" tab is not a control but a message box containing detailed information on the current selection. When users double-click connection lines, all the packets associated with that connection are displayed in this window. Double-clicking a host icon spawns a "whois" command to find information on the host's IP address. An example of this output appears in Figure 9.

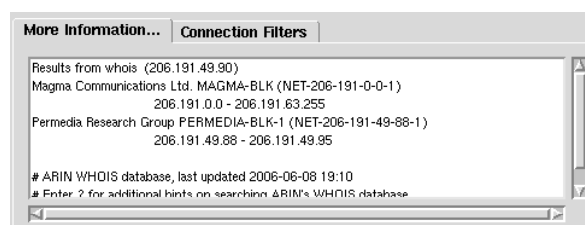


Figure 9: The More Information tab.

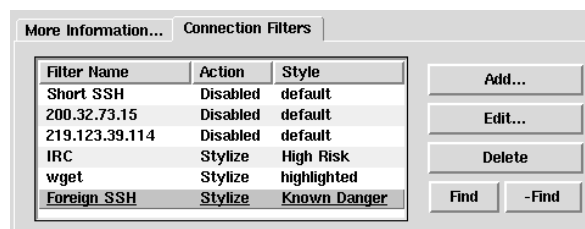


Figure 10: The Connection Filters tab.

Double-clicking on a port retrieves the information from the machine's /etc/services file that tells what protocols have registered the use of that port. In the future, this action might retrieve current information about what malicious programs are known to use that port number to communicate. Another future expansion of the "more information" tab would be to show what files a communicating process had open over its lifetime when the user double-clicks its icon. Our users expressly requested each of these "more information" features during the pilot study.

The second tab of this section contains connection filters (Figure 10). Filters are SQL expressions the

user can employ to remove extraneous display items from the detailed view showing only what is important. Filters may have other actions besides removing items from display. Filters may omit or stylize any display item (host, process, port, or connection) in the detailed view using the full matching power of SQL-92 queries. Additionally, one can use “filter-in” filters to display any items that would have been removed by a “filter-out” filter. This lets the user filter out, for instance, all Secure Shell traffic broadly, and then perhaps filter back in traffic that comes from a particular range of IP addresses. Users may also create a filter from something that is already displayed using a query-by-example technique.

An important use of filters is to highlight in all views the connections that either match or do not match the filter’s SQL. This capability allows users to quickly find features that match a search criterion. When the user selects a filter and presses the “Find” button, all connections that match the filter are highlighted in any views they appear in. Conversely, pressing the “-Find” button performs an inverse find, highlighting all the connections that do not match the SQL expression.

In practice, we found that users who employed the “Find” and “-Find” capabilities were able to hone in on an area of interest fastest. For instance, if a user believed that long-duration SSH connections were an indicator of a break-in, he might make a filter that contained the expression:

```
svPort = 22 AND (lastSeen - firstSeen) > 20
```

This expression would match all connections where the server port number was 22 (SSH) and the connection duration was greater than 20 seconds. The user could then find all connections that matched this criterion and zoom in on that area.

Users may define arbitrarily complex filters, but the best practice seems to be using a series of simple filters in succession. Filters are applied from top to bottom with later filters acting on the output of former filters. Thus a user may filter out all short SSH connections and then color all remaining SSH connections red with a separate filter. After filtering out whole classes of connections, a user may use a subsequent filter to restore a subset of the previously removed connections that match certain criteria.

The user may define filter to change the display of any matching item to any of eight predefined styles: Default, Highlighted, Safe, Low-Risk, Medium-Risk, High-Risk, Known Danger, and Unknown. The user may change colors, fonts, and line thicknesses for each of the predefined styles as desired. Additionally, users can style individual items manually, apart from filters, to mark items of interest. The intention is for users to highlight items of special interest with visual characteristics that are meaningful to them. We found the combination of powerful data filters and user-definable visual styles very useful for analysis.

Evaluation

Our objective for developing HoNe was to show that integrating host and network data into a single visual display provided useful insight for experienced system administrators trying to investigate anomalous behavior. After building HoNe, we rigorously evaluated it to find out how well our objectives were met and what remained to be done. We designed the evaluation to answer the following questions:

- Does visual packet-process correlation enhance intrusion-detection insight over tools currently in use?
- What are the benefits and pitfalls of visual vs. textual presentations?
- What are the benefits and pitfalls of packet-process correlated vs. noncorrelated presentations?

We were also curious to a lesser degree to find out what level of experience users needed to benefit most from HoNe. This section discusses the two-phase usability evaluation: pilot interviews and summative usability evaluation. The purpose of the pilot interviews was twofold: (1) to bring to light missing parts of the visualization, and (2) to determine how much intervention would be needed during the summative usability evaluation. The purpose of the summative usability evaluation was to quantitatively answer the research questions.

Pilot Interviews

We conducted a pilot study with six expert computer security professionals and two information visualization experts. We selected subjects based on their known expertise and their helpfulness in prior interviews. We asked the pilot study subjects to honestly evaluate the visualization while using it to discover a real hacking incident in data collected using our kernel modifications. Their valuable insight helped us determine what pieces were still missing from the visualization and how much intervention would be necessary for less experienced users in the study to be able to use HoNe productively.

From the pilot study, we learned that HoNe had great potential but needed a little refinement. There were 26 enhancement requests, six areas where intervention would be needed, eight negative comments, and 11 positive comments. We used this feedback to refine the user interactions and to add options to provide more information about hosts, processes, and ports as the experts requested.

We found that the area needing the most intervention was constructing SQL queries for filters. Our interviews made it clear that our initial implementation of time windows based on slider widgets was unusable, so we replaced it with a more graphical approach that employs histograms, connection lines, and direct manipulation. We added the “Find” and “Inverse Find” buttons to help users rapidly locate

information of interest within huge files. The pilot interviewees told us that HoNe would be an effective tool for intrusion detection with powerful filtering and graphical capabilities that would be most effective for discovering zero-day (previously unknown) exploits. They also told us that our correlated data was a new and unique source of information that could be used in automated intrusion prevention systems.

Summative Usability Experiment

We designed a two-factor experiment, with the primary factors being *visualization* – at two levels (on or off), and *correlation* – at two levels (on or off). The four experimental conditions were:

1. No-Visualization/No-Correlation (NVNC): User has no visualization and only the uncorrelated text output of tcpdump, netstat, and lsof to work from (Control condition).
2. No-Visualization/Correlation (NVC): User has a human-readable text version of the correlated data from the kernel modifications plus the control data.
3. Visualization/No-Correlation (VNC): User has separate visualization windows for netstat and tcpdump data. All the textual control data is available, but the correlated data is not.
4. Visualization/Correlation (VC): User has all the data from the previous conditions along with a visualization of the correlated data.

We collected data for four scenarios using our modified Linux kernel. For each scenario, we simultaneously collected data from netstat, lsof, tcpdump, syslog, and our kernel modifications. The netstat polling period was set to 1 sec, the lsof polling period was 60 sec, and all other data collection was continuous. The scenarios were each between 12 and 120 hours long. The following is a description of each scenario

1. Control: Linux RedHat Fedora Core 3 system running only an SSH server, no engineered attacks.
2. Engineered Hack: We hacked the machine by logging into an unprivileged account under cover of an SSH scan, downloading a rootkit from a remote server, and starting a new Internet service.
3. Normal: As control, but with attacks that do not result in intrusions.
4. Uncontrolled Hack: Similar to the engineered hack but real, not engineered by us.

There were 16 possible (condition, scenario) treatment combinations. We had each subject perform four runs under different treatment combinations such that each subject experienced every condition and scenario exactly once. We perturbed the order of treatment combinations to counter learning effects. Statisticians from an independent statistical consulting group verified that the experiment was balanced.

We recruited 27 subjects, about half of whom had more than a year of professional experience as a

system administrator. We provided help and training on all needed skills (e.g., writing SQL queries or tcpdump packet filters), and told each subject the limitations of how much we would help them. The intervention and training brought all the subjects up to a certain minimum skill level to allow them to complete the scenarios. We were interested not in how well people used basic tools such as tcpdump and grep, but in how well various viewing conditions helped them see what was happening in a dataset.

In each run, we asked the subjects to identify intrusions or any other security-related features of the data. We established our “ground truth” about scenario events using a priori knowledge and the judgment of security experts who were given all the captured data (including data not available to the experimental subjects) at their disposal. We assigned each security-relevant feature a unique identifier and a value using a seven-point scale (Table 3). Subjects gained points for noticing and correctly diagnosing features but lost points for incorrectly diagnosing a noticed feature. Subjects had approximately 15 minutes to diagnose features for each of their four runs.

Points	Meaning
-3	Diagnosing a benign feature as a malicious penetration or Missing a major malicious penetration
-2	Diagnosing a malicious nonpenetration as a malicious penetration or Missing an apparent penetration (given the condition) or Missing additional major malicious, nonpenetrating features beyond the first
-1	Diagnosing a benign feature as a malicious nonpenetration or Missing a major malicious nonpenetration
0	Noticing a benign feature
+1	Properly diagnosing a malicious nonpenetration or Properly diagnosing a malicious penetration without supporting reasoning (guessing)
+2	Properly diagnosing a malicious penetration
+3	Properly diagnosing and assessing the impact of a malicious feature (real or apparent) or Properly noting the relationship of two or more malicious features together

Table 3: Seven-point Insight Score scale.

For each scenario and viewing condition, we totaled the positive and negative scores separately. Larger absolute score values indicate more features were noticed. Higher positive scores indicate that the subject more often correctly diagnosed features he/she

noticed while higher (absolute) negative values indicate he/she tended to misdiagnose features. Scores closer to zero indicate fewer features were noticed.

We normalized the positive and negative scores by dividing the subjects' positive scores by the maximum possible score for the scenario used and dividing the negative scores by the minimum possible score. We used the normalized scores to compare scores across different scenarios. Then we subtracted the normalized negative score from the normalized positive score to obtain the insight score (Figure 11). We used the insight score to award prizes among other things. This approach prevents a user from attempting to increase his score by reporting lots of inconsequential features and reduces the likelihood of guessing. Accuracy is the most important characteristic of the kind of work we are studying. If administrators quickly reach an incorrect conclusion the consequences could be more costly than if they reach a correct conclusion more slowly than desired.

$$Insight = \frac{Noticed}{||Noticed||} - \frac{Misdiagnosed}{||Misdiagnosed||}$$

Figure 11: Formula for Insight Score.

Data Collection

We wanted interviewees to see real data from actual break-ins, so we set up a sacrificial host on the campus LAN and outfitted it with Snort and Tripwire (two freely available intrusion detection systems). Then we created a User Mode Linux (UML) [UML] virtual machine equipped with the modified kernel running in a process on the real machine. We bridged the real and virtual machines' Ethernet interfaces so that they looked like separate machines on the same LAN segment.

A defect in our kernel modifications caused the machine to crash when attackers attempted to subvert the machine in certain ways. Kernel-level rootkits such as Adore [Adore] and LRK5 [LRK5] caused the machine to crash when they were activated. Thus, after the attacker initially penetrated the machine, his attempts to install a rootkit would crash often the virtual machine. A positive result of this "feature" was that hackers were unable to use the machine to exploit other machines on the campus network. Unfortunately, the defect often prevented us from collecting data beyond the initial breakin.

Summative Study Findings

Users preferred the correlated visualization (VC) and felt they got the most insight from it of any of the conditions. The VC treatment resulted in better insight scores than the NVNC (control) at better than the 0.01 level of significance, and that the NVC treatment resulted in better scores than NVNC at the 0.1 level of significance. Additionally, we showed that VC was better than NVC with marginal statistical significance. From these findings, we infer that our visual correlation of packets to processes does help administrators

perform certain intrusion detection tasks better than text data alone could.

The VC condition garnered a large number of unsolicited positive comments about the visualization tool. We believe that users would have performed even better and had a more satisfying experience if not for some implementation bugs in the visualization and its suboptimal responsiveness.

The NVC condition provided the critical element of packet-process correlation without providing a means of visualizing the data. The correlated data was compact and contained in a single data file so users did not get lost nearly as easily as in the NVNC condition. Users who had high skills with text manipulation tools were often very adept at diagnosing problems in the NVC data. TCP conversation reconstruction (separating out individual pair-wise conversations from a larger set of many simultaneous conversations) was part of the correlation process that made reading packet traces easier for our users. The correlated data would have been much more useful to humans if we had put a start time and duration instead of start and end times that the users had to mentally subtract to get duration. NVC users mentioned many times that they were looking for long sessions in the text. Those who had performed a run with the visualization previously often said that they most missed the visual indication of duration.

The VNC condition turned out to be very troublesome both to users and to us. Users chafed at having two similar visualizations that told them slightly different aspects of the same data. Some of the confusion users had was due to the similar appearance but different meaning of the VC and VNC conditions. A better approach might have been to choose two separate visualizations, one tailored to displaying tcpdump data, and another tuned to show netstat data visually. In any case, it was more difficult than expected to test visualization and packet-process correlation as truly independent concepts.

Although some more experienced users did very well with the NVNC condition, it was the least preferred and resulted in the lowest scores on average. Most users found this condition confusing and error-prone. Novices performed very poorly with the staggering amount of data this condition presented. Many times novices would painstakingly try to understand each packet or connection attempt often covering no more than the first 1% of the data during the whole 15-minute run. We encouraged these users to look at other data, but many novices seemed to be unable to draw high-level, evidence-based conclusions, preferring instead to interpret small findings deep in the details.

The user's reactions highlight an important general function of visualization, especially for novice users: visualizations present information compactly, allowing users to think about the data globally. In fact, both visualization and packet-process correlation have the effect

of compacting and simplifying the data for human perception, although the effect of visualization seems to be more pronounced. In the future, other studies could be conducted in the user's actual work environments to see whether these conclusions hold in practice.

Conclusions

We have noted how the literature is divided between display of host data versus display of network data. We have shown that this disconnect is inherent to the TCP/IP networking model [RFC-791, RFC-793] and is codified into the kernels of all modern operating systems. Thus, automatic process-packet correlation has been impossible and was overlooked as an approach to security awareness in both literature and practice. But we have presented a solution that bridges the gap between the network and transport layers, enabling correlation of incoming packets with the processes that accept them. We have created the HoNe Visualizer as a user interface to this new correlated data and demonstrated that both experts and novices perform better on intrusion detection tasks with it than with text-based tools alone. Our research has advanced the science of computer security in several significant ways. To recapitulate what we believe are the most important contributions of our work:

- We have interviewed system administrators and identified areas where HCI research could improve their tools.
- We have identified the host/network divide, shown its causes, and examined its effects on computer security both technologically and cognitively.
- With HoNe, we have bridged the technical aspects of the host/network divide and laid the groundwork for bridging the cognitive aspects as well.
- We have created a visualization of packet-process correlation, making it possible for humans to make better diagnoses about the nature of connections.
- We have demonstrated the advantages of packet to process correlation via quantitative usability evaluations.
- We have created a new source of correlated data that will be useful to automated security monitoring tools as well as humans.
- We have generated new tools for system administrators via participatory design and performed usability evaluations to quantify their utility.

Future Plans

In the future we plan to replace the kernel modifications with a set of DTrace scripts. This will make gathering the correlated data safer and more portable. We plan on improving our visualization and making it more efficient, but other visualizations or back-end process could use this rich new source of data to increase the security of monitored machines and inform rapid and accurate responses to communications problems.

We believe our host/network bridge and visualization will be a valuable asset to system administrators. This type of visualization will amplify the insights and abilities of system administrators, complement the existing tools they use, reduce monitoring costs, and increase the security posture of organizations that use it.

Impact

HoNe has demonstrated how helpful packet-process correlation and visualization are for detecting and diagnosing potentially malicious activity on computers. We have also shown that the layered model, while effective in many ways, has problems caused by lack of visibility across software layers within the kernel. As Cantrill [Cantrill, 2006] noted, the main problem with layered systems of today is a profound lack of software observability. The only way to see what software is doing, especially system software, is to modify it. This is what we have done with HoNe: we modified the kernel to gain visibility into how packets relate to processes and created a visualization of the information we gathered. Since lack of observability is a huge problem in today's system software, we expect to see efforts such as DTrace [Cantrill, 2004] gain broader acceptance and become available on more platforms. But these programs only dump more text at the user. What we hope HoNe will show is how important it is to present computer security data to users in the way they can process it most rapidly, via visualization.

Much work remains to be done for system administrators, but it is our hope that HoNe will lay the groundwork for a positive change in the way security professionals go about their work. If kernel designers accept our conclusions and incorporate greater observability of the packet-process relationship into their work, much better security monitoring will be possible in the future than today. System administrators will be able to interrogate their systems for security problems more directly and then visualize the results. This kind of progress will make it much harder for malicious persons to hide their activities, making the entire Internet safer for its users.

Finally, we hope that the success of our work can demonstrate how important it is to talk to system administrators and involve them as co-designers in any work that purports to meet a need. Tools must adapt to their user's needs to be truly useful. The important thing is to find out what the users need (even if it is not what they actually asked for) and then design tools to fit the need. HoNe is one such tool—may many others follow it.

Acknowledgments

The authors would like to thank Virginia Tech's security officer and the system administrators we interviewed for their patience, enthusiasm, and guidance as we shaped this product to fit their needs. This research

was supported in part by a National Science Foundation Integrated Graduate Education and Research Training (IGERT) grant (award DGE-9987586).

Author Biographies

Dr. Chris North, Associate Professor of Computer Science at Virginia Polytechnic Institute and State University, is head of the Laboratory for Information Visualization and Evaluation and member of the Center for Human-Computer Interaction. He received his Ph.D. at the University of Maryland, College Park. He co-lead the establishment of Virginia Tech as an NSA National Center of Academic Excellence in Information Assurance Education. His current research interests are information visualization, high-resolution displays, and visualization evaluation methods. Contact at north@vt.edu, <http://infovis.cs.vt.edu/>.

Glenn Fink recently completed his Ph.D. at Virginia Tech where his dissertation was on visual correlation of network traffic and host processes for computer security. At this writing he is moving out to Washington state to accept a job at Pacific Northwest National Labs where he will work on more visualization technologies for computer security applications. Reach him via email at finkga@vt.edu.

Ricardo Correa received his BS in Computer Science from the University of Texas at El Paso. He worked for the University's IT department managing the campus-wide network infrastructure. He is currently pursuing an MS in Network Engineering at the University of Pennsylvania. Reach him electronically at ricm@seas.upenn.edu.

Vedavyas Duggirala is a currently pursuing his Ph.D at Virginia Tech. His work is in the area of large scale network emulation. He can be reached via vduggira@vt.edu.

References

- [Adore] Adore kernel-level rootkit, <http://www.packetstormsecurity.org/groups/teso/>, last accessed July 2006.
- [Cantrill, 2004] Cantrill, B., M. Shapiro, and A. Leventhal, "Dynamic Instrumentation of Production Systems," *Proceedings of the 2004 Usenix Annual Technical Conference*, 2004.
- [Cantrill, 2006] Cantrill, B., "Hidden in Plain Sight," *Queue*, Vol. 4, Num. 1, pp. 26-36, <http://doi.acm.org/10.1145/1117389.1117401>, Feb., 2006.
- [Card, et al., 1999] Card, S. K., J. D. Mackinlay, and B. Shneiderman, "Information Visualization," in Card, S. K., J. D. Mackinlay, and B. Shneiderman, eds. *Readings in information visualization: Using vision to think*, Morgan Kaufmann Publishers, San Francisco, Calif., pp. 1-34, 1999.
- [eHealth] eHealth, A network management tool owned by Computer Associates, Inc., http://www.concord.com/products/network_mgt.shtml, Last accessed July, 2006.
- [Fink, et al., 2005] Fink, G. A., P. Muessig, and C. North, *Visual Simultaneous Correlation of Host Processes and Network Traffic*.
- [Foundstone] Foundstone, Inc.'s free forensic tools, <http://www.foundstone.com/resources/freetools.htm>, Last accessed July, 2006.
- [Li and North, 2003] Li, Q. and C. North, "Empirical Comparison of Dynamic Query Sliders and Brushing Histograms," *Proceedings of IEEE Symposium on Information Visualization 2003*, pp. 147-154, 2003.
- [LRK5] LRK5 kernel-level rootkit, <http://packetstormsecurity.org/UNIX/penetration/rootkits/lrk5.src.tar.gz>, Last accessed July, 2006.
- [OSIArch] Zimmermann, Hubert, "OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection," *IEEE Transactions on Communications*, Vol. 28, Num. 4, pp. 425-432, April, 1980.
- [RFC-1122] RFC 1122, "Requirements for Internet Hosts – Communication Layers," <http://tools.ietf.org/html/rfc1122>, Last accessed July, 2006.
- [RFC-791] RFC 791, "Internet Protocol," September, 1981 (See also: MIL-STD-1777).
- [RFC-793] RFC 793, Transmission Control Protocol, September, 1981 (See also: MIL-STD-1778).
- [Sysinternals1] ProcessExplorer, <http://www.sysinternals.com/Utilities/ProcessExplorer.html>, Last accessed September, 2006.
- [Sysinternals2] TCPViewPro, <http://www.sysinternals.com/Utilities/TcpView.html>, Last accessed Sep., 2006.
- [UML] User Mode Linux, <http://user-mode-linux.sourceforge.net/>, Last accessed July, 2006.
- [ZoneAlarm] Zone Alarm Pro, Zone Labs, Inc., San Francisco, CA 94107, USA, 2003, <http://www.zonelabs.com/>, Last accessed August, 2005.

The NMI Build & Test Laboratory: Continuous Integration Framework for Distributed Computing Software

*Andrew Pavlo, Peter Couvares, Rebekah Gietzel, Anatoly Karp, Ian D. Alderman,
and Miron Livny – University of Wisconsin-Madison
Charles Bacon – Argonne National Laboratory*

ABSTRACT

We present a framework for building and testing software in a heterogeneous, multi-user, distributed computing environment. Unlike other systems for automated builds and tests, our framework is not tied to a specific developer tool, revision control system, or testing framework, and allows access to computing resources across administrative boundaries. Users define complex software building procedures for multiple platforms with simple semantics. The system balances the need to continually integrate software changes while still providing on-demand access for developers. Our key contributions in this paper are: (1) the development of design principles for distributed build-and-test systems, (2) a description of an implemented system that satisfies those principles, and (3) case studies on how this system is used in practice at two sites where large, multi-component systems are built and tested.

Introduction

Frequently building and testing software yields many benefits [10, 14, 17]. This process, known as *continuous integration*, allows developers to recognize and respond to problems in their applications as they are introduced, rather than be inundated with software bugs only when a production release is needed [2]. If the time span between the last successful build of an application and latest broken version is short, it is easier to isolate the source code modifications that caused the application's compilation or testing to fail [17]. It is important to fix these software bugs early in the development process, as the cost of the fix has been shown to be proportional to the age of the bug [3].

We developed the NMI Build & Test framework to facilitate automatic builds and tests of distributed computing software in a distributed computing environment. It is part of the NSF Middleware Initiative (NMI), whose mission is to develop an integrated national middleware infrastructure in support of science and engineering applications. In these types of problem domains, build-and-test facilities may be comprised of a few computing resources in a single location, or a large, heterogeneous collection of machines in different geographical and administrative domains. Our system abstracts the build-and-test procedures from the underlying technology needed to execute these procedures on multiple resources. The logically distinct steps to build or test each application may be encapsulated in separate, fully automated tasks in the framework without restricting users to any specific development tool or testing system. Thus, developers can migrate their existing build-and-test procedures easily without compromising the procedures of other applications using the framework.

To build and test any application, users explicitly define the execution workflow of build-and-test procedures, along with any external software dependencies and target platforms, using a lightweight declarative syntax. The NMI Build & Test software stores this information in a central repository to ensure every build or test is reproducible. When a build or test *routine* is submitted to the framework, the procedures are dynamically deployed to the appropriate computing resources for execution. Users can view the status of their routines as they execute on build-and-test resources. The framework captures any artifacts produced during this execution and automatically transfers them to a central repository. Authorized users can pause or remove their routines from the framework at any time.

We implement the NMI framework as a lightweight software layer that runs on top of the Condor high-throughput distributed batch computing system [15, 25]. Leveraging a feature-rich batch system like Condor provides our framework with the fault-tolerance, scheduling policies, accounting, and security it requires. The NMI Build & Test software is not installed persistently on all available computing resources; it is deployed dynamically by the batch system at runtime.

The framework and software that we present here are just one component of the NMI Build & Test Laboratory at the University of Wisconsin-Madison's Department of Computer Sciences. The Laboratory also provides maintenance and administration for a diverse collection of resources. It is used as the primary build-and-test environment for the Globus Toolkit [8] and the Condor batch system [25], as well as other products. Managing such a production facility

presents certain specific system administration problems, such as maintaining a standard set of software libraries across multiple platforms and coping with the large amount of data produced daily by our users.

In this document, we discuss not only the design and architecture of the NMI framework and software, but also the tools and practices we developed for managing a heterogeneous build-and-test facility on which it is deployed.

Related Work

Continuous integration and automated build-and-test systems are used by many large software projects [14]. The benefits of these systems are most often reported in discussions of agile software development [2, 12, 22]. Research literature on the general design of such systems, however, is limited.

There are numerous commercial and open source continuous integration and automated build systems available [13]. Almost all provide the basic functionality of managing build and test execution on one or more computing resources. Three popular systems are the Mozilla Project's Tinderbox system [20], the Apache Software Foundation's Maven [16], and the CruiseControl toolkit [6]. The Tinderbox system requires autonomous agents on build machines to continuously retrieve source code from a repository, compile the application, and send status reports back to a central server. This is different from the approach taken by Maven and CruiseControl, where a central manager pushes builds and tests to computing resources and then retrieves the results when they are complete.

Many systems make important assumptions about the scope and complexity of the computing environment in which they are deployed. For example, some require that all build-and-test resources be dedicated or that all users have equal access to them. Other systems assume that prerequisite software is predictably installed and configured by the system administrator on all machines in the pool. Users must hard-code paths to these external dependencies in their build-and-test scripts, making it difficult to reproduce past routines on platforms that have been patched or updated. Although these constraints may be appropriate for smaller projects with few users, they are not realistic for larger organizations with diverse administrative controls or projects involving developers located throughout the world.

Other systems offer more flexibility and control of the build-and-test execution environment. The ElectricCloud commercial distributed build system re-factors an application's Makefiles into parallel workloads executed on dedicated clusters [19]. A central manager synchronizes the system clocks for the pool to help ensure that a build script's time stamp-based dependencies work correctly. Another full-featured commercial offering is the BuildForge continuous

integration system [7]. It uses an integrated batch system to provide rudimentary opportunistic computing capabilities and resource controls based on user and group policies.

These systems seldom address the many problems inherent in managing workloads in a distributed environment, however. For example, a system must ensure that a running build or test can be cancelled and completely removed from a machine. This is often not an easy accomplishment; thorough testing of an application often requires additional services, such as a database server, to be launched along with the application and testing scripts may leave a myriad of files scattered about the local disk.

Motivation

In a distributed computing environment, a build-and-test system cannot assume central administrative control, or that its resources are dedicated or reliable. Therefore, we need a system that can safely execute routines on resources outside of one local administrative domain. This also means that our system cannot assume that each computing resource is installed with software needed by the framework, or configured identically.

Because of the arbitrary nature of how routines execute in this environment, non-dedicated remote computing resources are often less reliable than local build-and-test machines. But even large pools of dedicated resources begin to resemble opportunistic pools as their capacity increases, since hardware failure is inevitable. A routine may be evicted from its execution site at any time. We need a system that can restart a routine on another resource and only execute tasks that have not already completed. The build-and-test framework should also ensure that routines are never "lost" in the system when failures occur.

Lastly, we need a mechanism for describing the capabilities, constraints, and properties of heterogeneous resources in the pool. With this information, a system ensures that each build-and-test routine is matched with a machine providing the correct execution environment: a user may require that their build routines only execute on a computing resource with a specific software configuration. The system needs to schedule the routine on an available matching machine or defer execution until a machine satisfying the user's requirements becomes available. If a satisfying resource does not exist, the system needs to notify the user that their requirements cannot be met.

Design Principles

The NMI framework was designed in response to our experiences developing distributed computing software. Our first implementation was created to help merge the build-and-test procedures of two large software projects into a unified environment where they could share a single pool of computing resources and

be packaged into a single grid software distribution. Both projects already had different established practices for building and testing their applications using a menagerie of custom scripts and build tools. Thus, our goal was to develop a unified framework incorporating these application-specific scripts and processes.

We developed a set of design principles for distributed build-and-test systems to solve the problems that we encountered in this merging process. We incorporated these principles into our implementation of the NMI Build & Test system. They are applicable to other continuous integration frameworks, both large and small.

Tool Independent

The framework should not require a software project to use a particular set of development or testing tools. If the build-and-test procedures for an application are encapsulated in well-defined building blocks, then a clear separation of the blocks and the tools used to manipulate them permits diversity. In our system, users are provided with a general interface to the framework that is compatible with arbitrary build-and-test utilities. The abstraction afforded by this interface ensures that new application-specific scripts can be integrated without requiring modifications to, and thereby affecting the stability of, the framework or other applications.

Lightweight

The software should be small and portable. This approach has three advantages: (1) it is easier for system administrators to add new resources to a build-and-test pool, (2) users are able to access resources outside of their local administrative domain where they may be prohibited from installing persistent software, and (3) framework software upgrades are easier as only the submission hosts need to be updated.

The NMI Build & Test framework uses existing, proven tools to solve many difficult problems in automating a distributed computing workflow. Because it is designed to be lightweight, it is able to run on top of the Condor batch system and take advantage of the workload management and distributed computing features Condor offers. The NMI software only needs to be installed on submission hosts, where it is deployed dynamically to computing resources. By this we mean that a subset of the framework software is transferred to build-and-test resources and automatically deployed at runtime.

Explicit, Well-Controlled Environments

All external software dependencies and resource requirements for each routine must be explicitly defined. This helps to ensure a predictable, reproducible, and reliable execution environment for a build or test, even on unpredictable and unreliable computing resources.

When a routine's procedures are sent to a build-and-test resource for execution in the NMI system, the framework creates and isolates the proper execution environment on demand. The framework ensures that only the software required by the routine is available at run time. This may be accomplished in two ways: (1) the developer must declare all the external software that their application requires other than what exists in the default vendor installation of the operating system, or (2) the developer may use the framework interface to automatically retrieve, configure, and temporarily install external software in their routine's runtime environment.

Central Results Repository

A build-and-test system should capture all information and data generated by routines and store it in a central repository. It is important that system allows users to easily retrieve the latest version of applications and view the state of their builds and tests [10]. The repository maintains routine's provenance information and historical data, which can be used for statistical analysis of builds and tests.

The NMI framework stores the execution results, log files, and output created by routines, as well as all input data, environment information, and dependencies needed to reproduce the build or test. While a routine executes, the NMI Build & Test software continuously updates the central repository with the results of each procedure; users do not need to wait for a routine to finish before viewing its results. Any output files produced by builds or tests are automatically transferred back to the central repository.

Fault Tolerance

The framework must be resilient to errors and faults from arbitrary components in the system. This allows builds and tests to continue to execute even when a database server goes down or network connectivity is severed. If the NMI Build & Test software deployed on a computing resource is unable to communicate with the submission host, the routine executing on that resource continues unperturbed. When the submission host is available again, all queued information is sent back; routines never stop making forward progress because the framework was unable to store the results.

The framework also uses leases to track an active routine in the system. If the framework software is unable to communicate with a resource executing a routine, the routine is not restarted on another machine until its lease expires. Thus, there are never duplicate routines executing at the same time.

Platform-Independent vs. Specific

For multi-platform applications, users should be able to define platform-independent tasks that are only executed once per routine submission. This improves the overall throughput of a build-and-test pool. For

example, an application's documentation only needs to be generated once for all platforms.

Build/Test Separation

The output of a successful build can be used as the input to another build, or to a future test. Thus, users are able to break distinct operations into smaller steps and decouple build products from testing targets. As described above, the framework archives the results of every build and test. When these cached results are needed by another routine as an input, the framework automatically transfers the results and deploys it on the computing resource at run time.

NMI Software

We developed the NMI Build & Test Laboratory's continuous integration framework software based on the design principles described in the previous section. The primary focus of our framework is to enable software to be built and tested in a distributed batch computing environment. Our software provides a command-line execution mechanism that can be triggered by any arbitrary entity, such as the UNIX cron daemon or a source code repository monitor, or by users when they need to quickly build their software before committing changes [10]. We believe that it is important for the framework to accommodate diverse projects' existing development practices, rather than force the adoption of a small set of software.

The NMI framework allows users to submit builds and tests for an application on multiple resources from a single location. We use a batch system to provide all the network and workload management functionality. The batch system is installed on every machine in a build-and-test pool, but the NMI software is only installed on the submission hosts. The framework stores all information about executing routines in a central database. The output from routines is returned to the submission hosts, which can store them

on either a shared network storage system or an independent file system.

A build-and-test routine is composed of a set of *glue scripts* and a *specification file* containing information about how an application is built or tested. The glue scripts are user-provided, application-specific tasks that automate parts of the build-and-test process. These scripts together contain the steps needed to configure, compile, or deploy an application inside of the framework. The specification file tells the framework when to execute these glue scripts, which platforms to execute them on, how to retrieve input data, and what external software dependencies exist.

Workflow Stages

The execution steps of a framework submission are divided into four stages: fetch, pre-processing, platform, and post-processing (see Figure 1). The tasks in the pre- and post-processing stages can be distributed on multiple machines to balance the workload. A routine's results and output are automatically transferred to and stored on the machine that it was submitted from.

Fetch: In this stage, the framework retrieves all the input data needed to build or test an application. Instead of writing custom scripts, users declare where and how files are retrieved using templates provided by the framework. Input data may come from multiple sources, including source code repositories (cvs, svn), file servers (http, ftp), and the output results from previous builds. Thus, input templates document the provenance of all inputs and help ensure the repeatability of routines.

Pre-processing: This optional stage prepares the build-and-test routine for execution on computing resources. These tasks are often used to process the input data collected in the previous stage. The platform-independent tasks execute

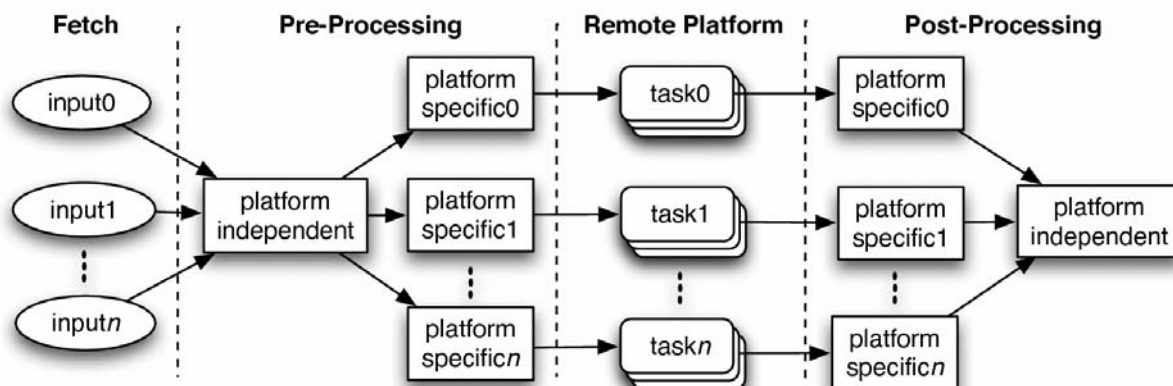


Figure 1: Workflow Stages – The steps to build or test an application in the NMI framework are divided into four stages. The fetch stage is executed on the machine that the user submitted the routine. The pre- and post-processing stages execute on any resource. The remote platform tasks each execute on the appropriate platform.

first and may modify the input data for all platforms. The framework then makes separate copies of the potentially modified input data for each platform and executes the platform-specific tasks. Any modifications made to the input data by the platform-specific tasks are only reflected in that platform’s copy.

Remote platform: After the input data is retrieved and processed, the framework submits one job for each target platform to the batch system. These jobs spawn the remote platform tasks to build or test an application on an appropriate compute resource. The NMI framework tells the batch system which input files to transfer to the resource along with a copy of the remote NMI framework software and the platform task glue scripts. Before these scripts begin to execute, the NMI software prepares the working directory for the routine and binds the execution environment paths to the local configuration of the machine. When each task finishes, any output produced can be sent back to the submission host for storage.

Post-processing: This stage contains tasks that process the output data produced by routines executing on build-and-test resources. As the platform tasks complete for each platform, the framework executes the platform-specific scripts for the corresponding set of results. Once these tasks are completed for all the platforms, the platform-independent scripts are then executed.

Workflow Manager

Using a distributed batch system to coordinate the execution of jobs running on the build-and-test machines provides the NMI framework with the robustness and reliability needed in a distributed computing environment.

We use the Directed Acyclic Graph Manager (DAGMan) to automate and control jobs submitted to the batch system by the NMI Build & Test software [5, 25]. DAGMan is a meta-scheduler service for executing multiple jobs in a batch system with dependencies in a declarative form; it monitors and schedules the jobs in a workflow. These workflows are expressed as directed graphs where each node of the graph denotes an atomic task and the directed edge indicates a dependency relationship between two adjacent nodes.

When a routine is submitted to the framework, its specification file is transformed into an execution graph. A single instance of DAGMan with this graph as its input is submitted to the batch system. DAGMan can then submit new jobs to the batch system using a standard application interface. As each of its spawned jobs complete, DAGMan is notified and can deploy additional jobs based on the dependencies in the graph.

DAGMan also provides the NMI Build & Test software with fault-tolerance. It is able to checkpoint a workflow much like a batch system is able to checkpoint a job. If the batch system fails and must be restarted, the workflow is restarted automatically and DAGMan only executes tasks that have not already completed.

Glue Scripts

A routine's glue scripts contain the procedures needed to build or test an application using the NMI framework. These scripts automate the typical human-operated steps so that builds and tests require no human intervention. Build glue scripts typically include configure, compile, and package steps. Test glue scripts can deploy additional services or sub-systems at runtime for thorough testing and can use any testing harness or framework.

The framework provides a glue script with information about the progress of its routine through predefined environment variables. Thus, the scripts can

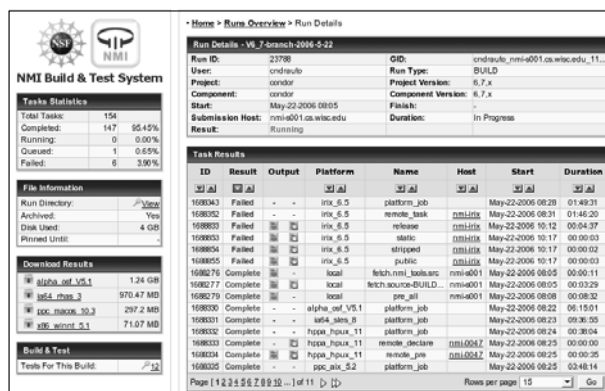


Figure 2(a): Routine status

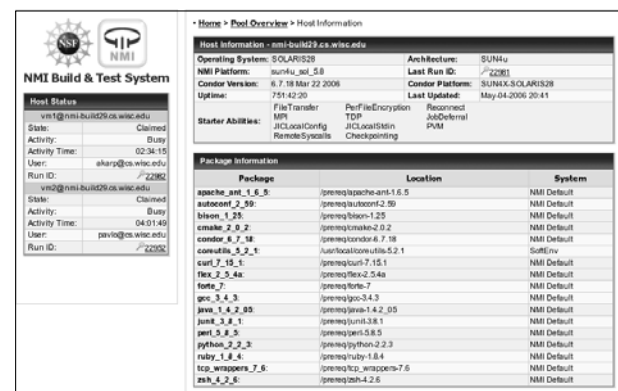


Figure 2(b): Computing resource information

Figure 2: NMI Framework Web Interface – The NMI Build & Test software provides a web client for users to view information about their build-and-test system. The screenshot in Figure 2(a) shows status information about a routine submitted to the framework; users can monitor the progress of tasks, download output files, and view log files. The screenshot in Figure 2(b) shows the capabilities of a machine, lists all prerequisite software installed, and provides information about the routines currently executing on it.

control a routine's execution workflow while they are running on a build-and-test resource. For example, a build glue script might halt execution if a dependency failed to compile in a previous step. Optionally, a test glue script may choose to continue even if the previous test case failed.

Application Interfaces

The NMI framework provides a standard interface for submitting and managing routines in a build-and-test system. This interface can easily be augmented by other clients or notification paradigms. For example, our framework distribution includes a web interface that provides an up-to-date overview of the system (Figure 2).

Batch System

We designed the NMI framework to run on top of the Condor high-throughput distributed computing batch system [15, 25]. When a user submits a build-and-test routine, the framework software deploys a single DAGMan job into Condor (Figure 3). This DAGMan job then spawns multiple Condor jobs for each platform targeted by the routine. Condor ensures that these jobs are reliably executed on computing resources that satisfy the explicit requirements of the routine.

Features

Condor provides many features that are necessary for a distributed continuous integration system like the NMI framework [24]. It would be possible to deploy the framework using a different batch system if the system implemented capabilities similar to the following found in Condor.

Matchmaking: Condor uses a central negotiator for planning and scheduling jobs for execution in a pool. Each machine provides the negotiator with a list of its capabilities, system properties,

pre-installed software, and current activity. Jobs waiting for execution also advertise their requirements that correspond to the information provided by the machines. After Condor collects this information from both parties, the negotiator pairs jobs with resources that mutually satisfy each other's requirements. The matched job and resource communicate directly with each other to negotiate further terms, and then the job is transferred by Condor to the machine for execution. The framework will warn users if they submit a build or test with a requirement that cannot be satisfied by any machine in the pool.

Fault tolerance: The failure of a single component in a Condor pool only affects those processes that deal directly with it. If a computing resource crashes while executing a build-and-test routine, Condor can either migrate the job to another machine or restart it when the resource returns. Condor uses a transient lease mechanism to ensure only a single instance of a job exists in a pool at any one time. If a computing resource is unable to communicate with the central negotiator when a job finishes execution, Condor transfers back the retained results once network connectivity is restored.

Grid resource access: Condor enables users to access computing resources in other pools outside of their local domain. Condor can submit jobs to grid resource middleware systems to allow builds and tests to execute on remote machines that may or may not be running Condor [11].

Resource control A long-standing philosophy of the Condor system is that the resource owner must always be in control of their resource, and set the terms of its use. Owners that are inconvenienced

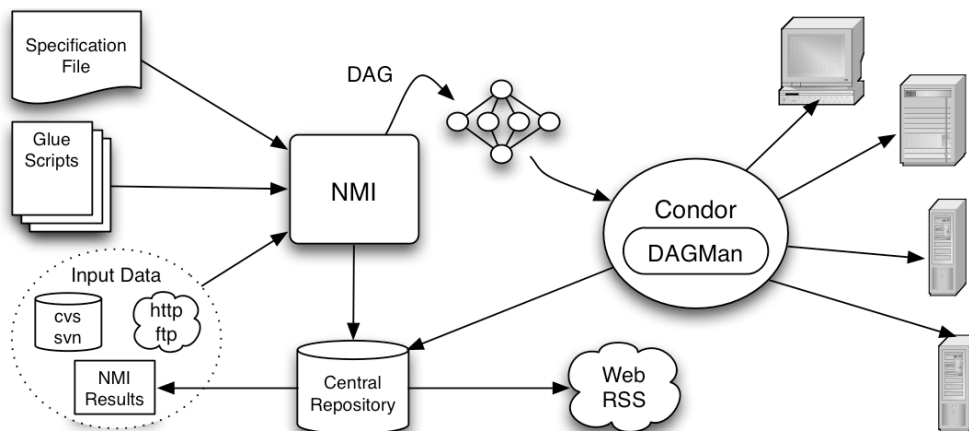


Figure 3: NMI Framework Architecture – The user submits a new routine comprised of glue scripts, input data, and a workflow specification file. The NMI software uses this information to create a dependency execution graph and submits a DAGMan job to the Condor batch system. When the DAGMan job begins to execute, it deploys multiple Condor jobs to the build-and-test computing resources. All output data produced by the routine's jobs are stored in a central repository and retrieved through ancillary clients.

by sharing their resources are less likely to continue participation in a distributed build-and-test pool. Condor provides flexible policy expressions that allow administrators to control which users can access resources, set preferences for certain routines over others, and limit when users are allowed to execute builds and tests.

Authentication: Condor supports several authentication methods for controlling access to remote computing resources, including GSI [9], Kerberos [23], and Microsoft's SSPI [1].

File transfer The NMI framework uses Condor's built-in file transfer protocol to send data between submission hosts and build-and-test resources. This robust mechanism ensures that files are reliably transferred; transfers are automatically restarted upon connection failure or file corruption. Condor can also use a number of encryption methods to securely transfer files without a shared file system.

Pool Configuration

Condor is designed to balance the needs and interests of resource owners, users wanting to execute jobs, and system administrators. In this spirit, Condor enables administrators to deploy and manage build-and-test pools that respect the wishes of resource owners but can still provide access for users. Priority schemes for both dedicated and non-dedicated resources can be created using Condor's flexible resource policy expressions. For example, the dedicated resources in a pool may prefer to execute processor-intensive builds and high-load stress tests so that shorter tests can be scheduled on idle workstations. Preferential job priority may also be granted to specific users and groups at different times based on deadlines and release schedules.

Condor can also further divide the resources of individual build-and-test machines, similar to the policies for the entire pool. Condor can allocate a multi-processor machine's resources disproportionately for each processor. For example, in one configuration a processor can be dedicated for build routines and therefore is allocated a larger portion of the system's memory. Test routines are only allowed to execute on the processor with more memory when no other jobs are waiting for execution. If a build is submitted while a test job is executing on this processor, Condor automatically evicts the test job and restarts it at a later time.

Build-and-test pools often have periods where there are no new routines available for execution. If a computing resource is idle for certain length of time, Condor can trigger a special task in the framework that performs continuous tests against an application as *backfill*. This is useful to perform long-term stress and random input tests on an application [18]. The results from these tests are reported back to the central repository periodically or whenever Condor evicts the backfill job to run a regular build or test routine.

Pool & Resources Management

We now discuss our experiences in managing the NMI Build & Test laboratory at the University of Wisconsin-Madison. The NMI framework is also currently deployed and running in production at other locations, including multi-national corporations and other academic institutions.

Operating System	Versions	Archs	CPUs
Debian Linux	1	1	2
Fedora Core Linux	4	2	20
FreeBSD	1	1	4
HP HP/UX	1	1	3
IBM AIX	2	1	6
Linux (Other)	3	2	9
Macintosh OS X	2	2	8
Microsoft Windows	1	2	3
OSF1	1	1	2
Red Hat Linux	3	2	13
Red Hat Enterprise Linux	2	3	19
Scientific Linux	3	2	11
SGI Irix	1	1	4
Sun Solaris	2	1	6
SuSE Enterprise Linux	3	3	15

Table 1: NMI Build & Test Laboratory Hardware – The laboratory supports multiple versions of operating systems on a wide variety of processor architectures.

Our facility currently maintains over 60 machines running a variety of operating systems (see Table 1). Over a dozen projects, representing many developers and institutions, use the NMI laboratory for building and testing grid and distributed computing software. In order to fully support the scientific community, we maintain multiple versions of operating systems on different architectures. Machines are not merely upgraded as newer versions of our supported platforms are released. We must instead install new hardware and maintain support for older platform combinations for as long they are needed by users.

Resource Configuration

We automate all persistent software installations and system configuration changes on every machine in our build-and-test pool. Anything that must be installed, configured, or changed after the default vendor installation of the operating system is completely scripted, and then performed using cfengine [4]. This includes installing vendor patches and updates. Thus, new machine installations can be added to the facility without requiring staff to rediscover or repeat modifications that were made to previous instances of the platform.

Prerequisite Software

In a multi-user build-and-test environment, projects often require overlapping sets of external software and libraries for compilation and testing. The NMI framework lets administrators offer prerequisite software for

routines in two ways: (1) the external software can be pre-installed on each computing resource and published to the NMI system, or (2) the system can maintain a cache of pre-compiled software to be deployed dynamically when requested by a user. Dynamic deployment is advantageous in environments where routines may execute on resources outside of one administrative domain and are unable to expect predictable prerequisite software.

At the NMI Laboratory, we use cfengine to install a large set of prerequisite software on each of our computing resources. This eases the burden on new users whose builds expect a precise set of non-standard tools but are not prepared to bring them along themselves. The trade-off, however, is that these builds and tests are less portable across administrative domains.

Data Management

The NMI Laboratory produces approximately 150 GB of data per day. To help manage the large amount of data generated by builds and tests, the framework provides tools and options for administrators.

Multiple submission points: More than one machine can be deployed as a submission host in a build-and-test pool. By default, the output of a routine is archived on the machine it is submitted from. The framework provides a built-in mechanism to make these files accessible from any submission host without requiring users to know which machine the data resides on. If a user requests output files from a previous build on a different submission host, the framework automatically transfers the files from the correct location.

Repository pruning The framework provides mechanisms for removing older build and test results from the repository based on flexible policies defined by the lab administrator. When the framework is installed on a submission host it deploys a special job into the batch system that periodically removes files based on the administrator's policy. Routines may be pruned based on file size, submission date, or other more complicated properties, such as duplicate failures. This process will only remove user-specified results; task output log files, error log files, and input data are retained so that builds and tests are reproducible. Users can set a routine's preservation time stamp to prevent their files from being removed before a certain date.

Case Studies

The NMI Laboratory is used as a build and test facility for two large distributed computing research projects: the Globus Toolkit from the Globus Alliance [8], and the Condor batch system from the University of Wisconsin-Madison's Department of Computer Sciences [15]. We present two brief case studies on how the NMI framework has improved each of these projects software development process.

Globus Toolkit

The Globus Toolkit is an open source software distribution that provides components for constructing large grid systems and applications [8]. It enables users to share computing power and other resources across administrative boundaries without sacrificing local autonomy. Globus-based systems are deployed all across the world and are the backbone of many large academic projects and collaborations.

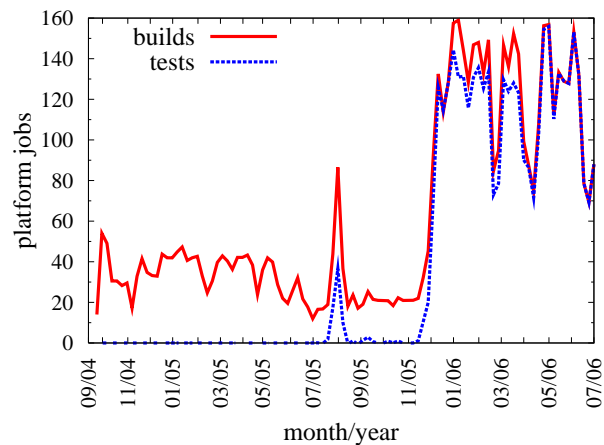


Figure 4: Globus Builds & Tests – The large spike in the number of jobs in the graph indicates when a new version of Globus was released and required many new build and test routines. Initially, the toolkit's build-and-test procedures were contained in a monolithic batch script. The tests were then later broken out of the build scripts into separate tasks. Thus, no data exists on these tests that were executed in the first months after switching to the NMI system.

Prior to switching to the NMI framework, the Globus system was built and tested using a combination of custom scripts and the Tinderbox open-source continuous integration system [20]. Each build machine contained a pre-defined source file that mapped all the external software needed by the build process to paths on the local disk. This file contained the only record in the system of what external software was used to execute a build or test, and did not contain full information about the specific version used. If the computing resource was updated to use a newer version of the software, there was no record in the build system to reflect that fact.

As the project grew, developers received an increased amount of bug reports from users. Many of these reports were for esoteric platforms that were not readily available to the Globus developers. Fewer builds and tests were submitted to these machines, which in turn caused bugs and errors to be discovered much later after they were introduced into the source code.

Now the Globus Toolkit is built and thoroughly tested every night by the NMI Build & Test software

on 10 different platforms (Figure 4). The component glue scripts for Globus contain the same build procedures that an end-user follows in order to compile the toolkit. These procedures also include integrity checks that warn developers when the build process generates files that are different from what the system expected. All other regression and unit tests are preformed immediately after compilation. Globus' developers have benefited from the NMI framework's strict attention to the set of software installed on computing resources and its ability to maintain a consistent execution environment for each build-and-test run. This allows them to test backwards compatibility of their build procedures with older versions of development tools, which they were unable to do before.

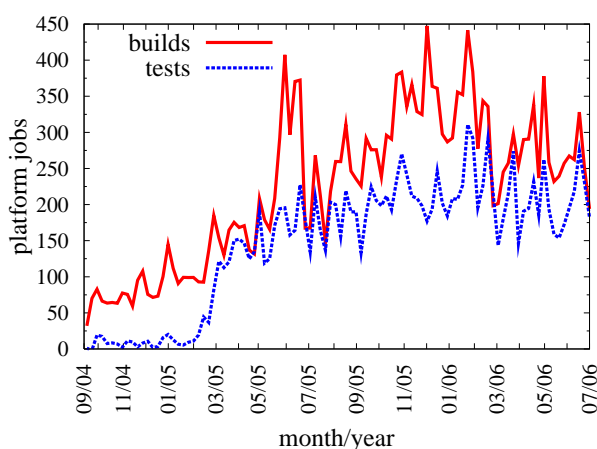


Figure 5: Condor Builds & Tests – Each platform job is a single build or test execution cycle on a computing resource; there may be multiple platform jobs for a single framework build-and-test routine. Sharp increases in the number of builds correspond to release deadlines for developers.

Condor

Before the advent of Linux's popularity, Condor supported a modest number of operating systems used by the academic and corporate communities. Initially, each developer was assigned a platform to manually execute builds and given a paper checklist of tests to perform whenever a new production release was needed. All of Condor's build scripts contained hard-coded path information for each machine that it was built on. If one of these machines needed to be rebuilt or replaced, the administrator would have to construct the system to exactly match the expected specification.

Like Globus, the Condor development team also deployed a Tinderbox system to automate builds and tests on all the platforms that were supported. Due to hardware and storage limitations, however, this system could only build either the stable branch or the development branch of Condor each day; developers had to make a decision on which branch the system should build next. This also meant that the system could not

easily build custom branches or on-demand builds of developer's workspaces.

Since transitioning to the NMI framework, the Condor project has experienced a steady increase in the number of builds and tests (Figure 5). The development team submits an automatic build and test to the framework every night for both the stable and development releases; Condor is built on 17 platforms with 122 unit and regression tests per platform. In addition, the framework is used for numerous on-demand builds of Condor submitted by individual developers and researchers to test and debug experimental features and new platforms.

Future Work

Many facets of the NMI framework can be expanded to further improve its capabilities.

Currently, the NMI framework coordinates builds and tests on multiple platforms independently. Each routine executes on a single computing resource for each specified platform. We are developing a mechanism whereby a build-and-test routine can execute on multiple machines in parallel and allow them to communicate with one another. Users specify an arbitrary number of machines and the batch system deploys the routine only when it has simultaneous access to all of the resources it requires. The framework passes information to the glue scripts about which machines are running the other parallel instances of the routine. Such dynamic cross-machine testing will allow users to easily test platform and version interoperability without maintaining permanent "target" machines for testing.

We are also extending our test network into the Schooner [21] system, based on Emulab [26], to expand these distributed tests to cover a variety of network scenarios. Schooner permits users to perform tests which include explicit network configurations. For example, the NMI framework will be able to include automated tests of how a distributed application performs in the presence of loss or delay in the network. This system will also allow administrators to rapidly deploy a variety of different operating system configurations both on bare hardware and in virtual machines.

A major boon to the NMI framework will be the proliferation of virtualization technology in more systems. Instead of deploying and maintaining a specific computing resource for every supported platform, the framework would keep a cache of virtual machine images that would be dynamically deployed at a user's request. Because administrators will only need to configure a single virtual machine image for each operating system in the entire pool, this will simplify build-and-test pool management and utilization. The framework would then also be able to support application testing that requires privileged system access or which makes irreversible alterations to the system configuration; these

changes would be localized to that instance of the virtual operating system and not the cached image.

Availability

The NMI Build & Test Laboratory continuous integration framework is available for download at our website under a BSD-like license: <http://nmi.cs.wisc.edu/>.

Acknowledgments

This research is supported in part by NSF Grants No. ANI-0330634, No. ANI-0330685, and No. ANI-0330670.

Conclusion

We have presented the NMI Build & Test Laboratory continuous integration framework software. Our implementation is predicated on design principles that we have established for distributed build-and-test systems. The key features that distinguish our system are (1) its ability to execute builds and tests on computing resources spanning administrative boundaries, (2) it is deployed dynamically on heterogeneous resources, and (3) it maintains a balance between continuous integration practices and on-demand access to builds and tests. Our software uses the Condor batch system to provide the capabilities necessary to operate in a distributed computing environment. We discussed our experiences in managing a diverse, heterogeneous build-and-test facility and showed how the NMI framework functions as the primary build-and-test system for two large software projects. From this, we believe that our system can be used to improve the development process of software in a distributed computing environment.

Author Biographies

Andrew Pavlo, Peter Couvares, Rebekah Gietzel, and Anatoly Karp are members of the Condor research project at the University of Wisconsin-Madison's Department of Computer Sciences. Ian D. Alderman is a Ph.D. candidate at the University of Wisconsin-Madison's Department of Computer Sciences. Miron Livny is a Professor with the Department of Computer Sciences at the University of Wisconsin-Madison and currently leads the Condor research project.

Charles Bacon is a researcher specializing in grid technology at Argonne National Laboratory.

Bibliography

- [1] *The security support provider interface*, White paper, Microsoft Corp., Redmond, WA, 1999.
- [2] Beck, K., *Extreme programming explained: embrace change*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
- [3] Boehm, B. W. and P. N. Papaccio, "Understanding and controlling software costs," *IEEE Transactions Software Engineering* 14, Vol. 10, pp. 1462-1477, 1988.
- [4] Burgess, M., "A site configuration engine," *USENIX Computing Systems*, Vol. 8, Num. 2, pp. 309-337, 1995.
- [5] Couvares, P., T. Kosar, A. Roy, J. Weber, and K. Wenger, *Workflows for e-Science*, Chapter: Workflow Management in Condor, Springer-Verlag, 2006.
- [6] *CruiseControl*, <http://cruisecontrol.sourceforge.net>.
- [7] Fierro, D., *Process automation solutions for software development: The BuildForge solution*, White paper, BuildForge, Inc., Austin, TX, March, 2006.
- [8] Foster, I., and C. Kesselman, "Globus: A meta-computing infrastructure toolkit," *The International Journal of Supercomputer Applications and High Performance Computing*, Vol. 11, Num. 2, pp. 115-128, Summer, 1997.
- [9] Foster, I. T., C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids," *ACM Conference on Computer and Communications Security*, pp. 83-92, 1998.
- [10] Fowler, M., *Continuous integration*, May, 2006, <http://www.martinfowler.com/articles/continuousIntegration.html>.
- [11] Frey, J., T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A computation management agent for multi-institutional grids," *Cluster Computing*, Vol. 5, pp. 237-246, 2002.
- [12] Grenning, J., "Launching extreme programming at a process-intensive company," *IEEE Software*, Vol. 18, Num. 6, pp. 27-33, 2001.
- [13] Hellesoy, A., *Continuous integration server feature matrix*, May, 2006, <http://damagecontrol.codehaus.org/Continuous+Integration+Server+Feature+Matrix>.
- [14] Holck, J., and N. Jørgensen, "Continuous integration and quality assurance: A case study of two open source projects," *Australian Journal of Information Systems*, Num. 11/12, pp. 40-53, 2004.
- [15] Litzkow, M., M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," *Proceedings of the 8th International Conference of Distributed Computing Systems*, June, 1988.
- [16] *Apache Maven*, <http://maven.apache.org>.
- [17] McConnell, S., "Daily build and smoke test," *IEEE Software*, Vol. 13, Num. 4, p. 144, 1996.
- [18] Miller, B. P., L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Communications of the Association for Computing Machinery*, Vol. 33, Num. 12, pp. 32-44, 1990.
- [19] Ousterhout, J., and J. Graham-Cumming, *Scalable software build accelerator: Faster, more accurate builds*, White paper, Electric Cloud, Inc., Mountain View, CA, February, 2006.
- [20] Reis, C. R., and R. P. de Mattos Fortes, "An overview of the software engineering process

- and tools in the Mozilla Project,” *Workshop on Open Source Software Development*, Newcastle, UK, pp. 162-182, 2002.
- [21] Schooner, <http://www.schooner.wail.wisc.edu>.
- [22] Schuh, P., “Recovery, redemption, and extreme programming,” *IEEE Software*, Vol. 18, Num. 6, pp. 34-41, 2001.
- [23] Steiner, J. G., B. C. Neuman, and J. I. Schiller, “Kerberos: An authentication service for open network systems,” *Proceedings of the USENIX Winter 1988 Technical Conference*, USENIX Association Berkeley, CA, pp. 191-202, 1988.
- [24] Tannenbaum, T., D. Wright, K. Miller, and M. Livny, “Condor – a distributed job scheduler,” *Beowulf Cluster Computing with Linux*, T. Sterling, Ed., MIT Press, October, 2001.
- [25] Thain, D., T. Tannenbaum, and M. Livny, “Distributed computing in practice: the condor experience,” *Concurrency – Practice and Experience*, Vol. 17, Num. 2-4, pp. 323-356, 2005.
- [26] White, B., J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, “An integrated experimental environment for distributed systems and networks,” *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, pp. 255-270, USENIX Association, Dec., 2002.

Unifying Unified Voice Messaging

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Roughly 18 months after installing a unified voice messaging system, we picked it up and merged it into our pre-existing production email domain. This paper deals with both technical aspects deploying a unified messaging system, as well as the cultural shock of merging the very different operational domains of Email support with Telecommunications support into a shared support model. As an added bonus we will discuss the merging two Exchange/Active Directory worlds into one with minimal impact on the existing users of both systems. Lastly, we will discuss some issues of operating a partially unified voice messaging system.

Introduction

After 12 years of service, our campus voicemail system failed. We were finally able to restore service after a 10 day¹ outage, but it was now clear to everyone, that this system needed to be replaced. Although we had been evaluating replacement voicemail systems prior to the failure, this changed the process from a theoretical exercise into a crash product selection and deployment exercise. Several years earlier [3], our division had implemented a Microsoft Exchange email service and this was in use by about 700, mostly administrative users. The eventual proposal was to install a Unified voice messaging system from Cisco that uses Exchange as a back end message store.

One of the attractive features of this product was unified voice messaging – that is your voicemails would appear as emails with .WAV files attached in your inbox. You could listen to them via the telephone like normal voicemail, or you could “play” the message via your email client. Since they were email messages, you could also use your email client to file or forward the message to other people. You could also get the system to “read” your email to you over the telephone (although good spam filtering is helpful here). Another benefit is that the new system could join the existing Windows 2000 domain and use the existing exchange servers. This allows us to leverage both existing hardware and existing support staff. The proposal went forward to the President’s cabinet and emergency funding was allocated.

Unfortunately, no one had actually talked with folks who supported the existing Exchange installation and we received some push-back from them when we mentioned making changes to the Active Directory schema and creating another 5000 mailboxes to go along with the 700 they currently supported, as well as adding this new service with full administrator rights to the domain. So instead, we found a few more servers and installed a second Windows 2000 domain with it’s own Exchange server. Although this allowed us to cut over from the creaky old voicemail system so

we had new hardware, it created some new problems for us. The first was that we now had to support a number of windows machines, along with a our own exchange server, and between our PBX Switch engineers and our Network Engineers, we were not in a happy place [6]. The second, and perhaps bigger problem, is that we had promised to deliver Unified Messaging, not just a replacement for the old, telephone only interface, voicemail system. In order to deliver on this promise, we had to find a way to merge the two domains. We also wanted to avoid another hard cut over like the initial installation, where all of our users had to set up their mailboxes and record greetings but then lost all of their existing messages.

Initial Installation

Our initial installation was a replacement of our Octel voice messaging system with the Cisco Unity voicemail system. Since we were setting up a stand alone Exchange domain for this system, we did not have to worry about unified messaging at this point, just plain old voicemail accessed via a telephone set. At that time, we had an Intecom S80 PBX and a few experimental (for us) Voice Over IP (VOIP) phones. The PBX communicated with the voicemail system using a bunch of analog telephone lines and a single RS232 serial line for control and switching information. This single interface would prove to be a major factor in how we managed the switchover. Since the PBX only understood (and could communicate with) a single voicemail system, we had to do a hard switchover; bring up the new system, move the wires from old to new, and shut off the old system.

The Unity product includes a web based tool for administration, as well as several bulk load tools to process CSV files. Through these tools, an administrator could create and destroy subscribers (which includes the exchange mailbox). We had two issues with this approach; the first is that a GUI interface is just too much work to manage ongoing changes,² and the

²We had previously automated the creating and expiration of student voicemail assignments based on room assignments in the Housing Office database.

¹Some replacement parts were obtained via eBay

second is that we would have the creation and deletion of Exchange accounts in the hands of Telecom staff³ instead of the **Exchange Administrator**. We needed a way to give telecom staff the ability to add and delete voicemail boxes, and we wanted to automate and integrate with other systems as much as possible. Although the Cisco interfaces had some provisions to enable individual users to do some of their own maintenance, we also wanted the ability of departmental administrators to manage the people in their departments. We have found the departmental administrator concept a very useful and powerful thing on our directory (white pages) deployment [2] and we wanted to be able to use that with the voice mail system.

In our deployment of our original windows domain, we had automated a good deal of our data flow [3] and were comfortable with that concept. But in order to enable the distributed control and management that we wanted, we had to strictly enforce our policy and practices. To do this, we really needed to replace the Cisco tools with our own tools. To this end, we wrote our own web tool to allow the management of mailboxes and call handlers, and enforce our naming conventions and additional record keeping along the way. One of our new rules, is that all mailboxes need to be “owned” by either a department or an individual, thus the new tool can display all of the mailboxes (and call handlers) owned by a specific person or department. In fact, the only way to create a new mailbox or call handler, is to first select the owner, and from there, creation can continue. The tool also attempts to avoid errors and do as much of the “thinking” as possible. When a new mailbox is created, the tool first checks to see if that number (telephone extension) is already assigned to another mailbox. If it is in use, it will note that fact, and give several options such as expiring the existing mailbox which would free up the number, or creating an Enhanced Call Processor.⁴ Creating the ECP automatically and then assigning the original mailbox to “1” and the new mailbox to “2” saved a few steps for the telecom staff and helped avoid procedural errors. This would also generate a “work item.” In the initial deployment, a staff member would still need to record the greeting (“Press one for Sam, press two for John”).

One of the tasks our telecom staff dreaded at the start of the semester, was setting up all of the student ECPs. They would have to record greetings for over 800 call handlers (“press one for Marquia, press two for Sharon”). We made two changes to this process. The first was to use some Text to Speech routines to preset the subscribers name when we created a mailbox. We were actually pretty pleased with how well

³It isn't that we don't trust our Telecom staff, but they do not have the training to administer Exchange and they are not in the department that is responsible for the Exchange service.

⁴ECP – A TLA we inherited from the Octel voicemail system – press 1 for John, press 2 for Sam

these routines did with many of the names. The second change was when we set up the call handler, we would generate the greeting by concatenating the names for each mailbox with the appropriate “press xxx for” phrases. This has the additional feature that if the subscriber records their own name, when we regenerate the greeting for the call handler, it will use the subscribers name recorded in their own voice.

We had previously written a tool to manage student voice mail accounts. This tool would look at the room assignments from residence life, and automatically assign mailboxes based on the room (which determined the phone number.) In the earlier version, this would generate a CSV file that would be loaded into the Octel system. This tool was modified to create the mailboxes and populate the appropriate call handlers. In this case, there were two types of handlers, a “Direct Transfer to xxxx” for single rooms and “Student Residence ECP xxxx” for shared rooms. Each student mailbox was assigned a six digit “directory number.”⁵ By using a call handler to access all student mailboxes, we did not need to worry about changing the extension assigned to mailbox, and as students changed rooms, they could keep the same six digit directory number, even as their room number (and phone number in their room) changed.

Subscribers, Call Handlers and Distribution Lists, Oh My!

A detailed discussion of the Unity Voice Messaging system would take a few days, and during our deployment, we purchased a web based training package to help jump start the Unified Messaging Team. But a brief overview of how Unity works would be helpful. The core building block of the Unity system is the call handler. This defines how a call is processed, what messages are played, what options the caller has during the call, and what happens after the greeting has been played.

Each call handler has some basic attributes including a name, an owner (which is a unity subscriber or unity distribution list) and an optional recorded name and an optional extension. It can also have a number of different types of greetings; a standard greeting, a “closed” greeting to be used when the university is closed, and alternate greeting, a busy greeting and an internal greeting. Each of these greetings can be enabled or disabled (the standard is always enabled), a source for the greeting which can be a recorded message (stored in a .WAV file, a system greeting (if there is a recorded name, it will be “*RECORDED NAME* is busy or unavailable” otherwise just “Extension XXXX is busy or unavailable”) or simply blank. There are some switches to control if caller input is accepted during the greeting, how many times to replay the message and finally, what to do

⁵Our campus uses four digit dialing – these six digit numbers are not directly dial-able, but can be dialed from within the voicemail system.

after the greeting is played. These options include taking a message, transferring to another subscriber or call handler, transferring to another extension or simply hanging up.

Call handlers can also accept caller input – a touch tone key press that can transfer to another subscriber – or any of the options for after playing the greeting. This can be configured for each of the 12 touch tone key presses. You can also configure how long a message can be left, who the message goes to, and what happens after the message is recorded. There are more options and settings available, but we are not currently using all of them.

The next part of the Unity system is the subscriber. This is a call handler with a message store attached, and some user information such as a PIN, a display name for the directory (both Active Directory and some directory lookup functions in Unity) and the location of the mail store. In general, the messages are stored in an Exchange server. This allows subscribers to access the messages via Outlook, or via the telephone (The Unity system is also an Exchange user that can open any mailbox). There is an option for messages to be sent immediately via SMTP to some other mail server. Although this means they can't access their voicemail via the telephone, since it is out of Unity's reach, they do get it via their preferred email client. This option is attractive to some of our users who don't use Exchange.

Unlike stand alone call handlers, subscribers (or their call handlers) must have an extension. Subscribers can also have alternate extensions, which allows a person with more than one phone line to have all of their calls go to one voice mailbox. Another attribute of subscribers, is notification and the message waiting indicator (MWI). The MWI is set when there are "unread" voice messages in their inbox.⁶ The system can also notify you of voicemail (and emails if you wish) via phone calls, pagers or email notification.

Another part of the Unity system, are distribution lists. These can be used to send group messages to folks and they can also be the owner of call handlers. We have two categories of automatically maintained lists, an "All Hands XXXX" list for some departments who request it, and "department admin XXXX" list with the department administrators for any department that owns call handlers. These are created automatically when a call handler is created. The owner of a call handler can update the greeting, this allows several folks in a department to maintain the greetings easily.

Although Unity has an "owner" for each object, we maintain our own owner information in our Oracle

⁶There is a childlike attraction to selecting "mark as unread" in Outlook and having the red light on your phone turn on.

database which allows us to delegate control and administration more effectively.

Functional Layout

In Figure 1, we have a diagram of our configuration. We have our old legacy phone system (PBX), which is connected to the voicemail system via both analog lines for voice traffic, and a serial line for control information. This PBX is also connected to our Call manager to provide a path for VOIP phones to talk with PBX based telephones. Both the Call manager and the PBX are have trunks to connect to our regular phone carriers for inbound and outbound phone service.

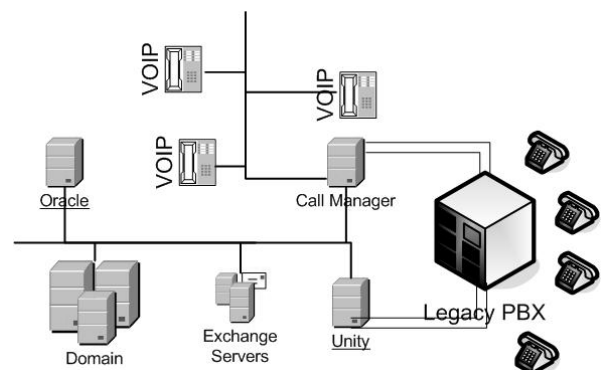


Figure 1: Function Diagram.

The call manager and the voicemail system (Unity) are also connected to the campus network, albeit behind firewalls. They use the network to communicate with each other, and Unity also connects to the Exchange message store. The call manager also uses the network to communicate with the individual voip telephones. The configuration (user provisioning at least) of the active directory domain, the call manager and the Unity system are all driven by an Oracle database. This database also handles our Identity Management functions and also drives our LDAP directory databases and many other aspects of our operation.

Write Only Memory

In our initial version of Unity, Cisco offered a C# API [7], so we built a process (the Unity Queue Runner) that would run on the Unity server, accept commands via a queue in our database [5] and process them via the API. We were pressed for time on this installation (recall that the original system had already failed hard for 10 days and was showing signs of additional failures) and this command path was one way only. We had to issue commands and assume that they worked. We kept track of what we thought we had done, and hoped that we were right.⁷ At this point, we had no way to read back the contents of the configuration database on the Unity server.

⁷This actually worked quite well, after 18 months of operation, we had only had a handful of errors – all due to human intervention

What's In A Name

We ran into a few issues with our naming conventions. One of the first problems we had, is that when the Unity Queue Runner was passing commands to Unity, it would reference the mailbox or call handler by its name. On a few occasions, someone using the Cisco tool, had changed the name directly in Unity. As a result, the command would fail – since the object wouldn't be found. Although there were only a few of them, they generated quite a few headaches for the technical staff, as well as problems for the front office folks. Eventually we were able to convince the front office folks that “never change a name,” meant exactly that. We actually could go into the SQLServer database on Unity to find and fix them, but it was very annoying.

A more common problem, and very annoying, is that we used names like ‘Fac/Staff ECP nnnn’ or ‘Student ECP nnnn’ for our call handlers. What we did not know at the time, that the call handler search function on the Cisco tool only used the first eight characters, and there was no way to search by extension. Thus, the search tool would return a list of all ‘Fac/Staff ECPs,’ an unsorted list at that. This did have the minor benefit of making the Cisco tool almost useless for working with call handlers, which forced the front office staff to use our own tools and reduced the first problem a bit.

Another part of our early naming convention, was to embed the department name or abbreviation into the name of call handlers for departments (ECPs and Voicemail trees). While this made some degree of sense for the Octel system (where we inherited the practice), it proved to have several issues with Unity. In the original deployment, we ran into departments that did not have abbreviations, and the resulting name was too long. Even with the abbreviations, the search problem mentioned above hit us just about every time. The other issue that we ran into, is that departments changed their name and abbreviations periodically, which then meant that we had to rename call handlers, which we can't easily do.

Since our tools to manage the Unity system had additional fields to capture owner information (and whatever else we wanted), we no longer needed to overload the call handler names like we did with the Octel system. Some of these naming problems we were able to correct before our second upgrade; we were able to destroy and recreate all student call handlers over the semester break. In other cases, we kept the old (longer) names and new call handlers would get the shorter names following the new convention.

Octel to Unity Switchover

The migration from the old Octel system to the Cisco Unity system was in some respects easier. People understood that the Octel system was about ready to die, and had lower expectations. On the other hand, there was a lot more busy work required. We were able to get a dump of the Octel configuration, which included the directory number, class of service, the

owner “name” (this was a free text field, with no validation) and some other configuration information. We needed to map this information on to the new system, and assign owners to everything.

The student voicemail was pretty easy – we were already generating those assignments in Oracle and loading them into the Octel; we wrote some interface code, and that process could talk to Unity. That left the fac/staff mailboxes to be migrated. We wrote a tool that helped us assign owners to Octel mailboxes (and save this info in Oracle). Some of this could be done automatically. If an Octel mailbox (directory number) was listed exactly once in the telephone directory, and that person was an active employee, we made the match. The rest needed to be cleaned up manually. The tool would display unmapped mailboxes, and the front office staff could work through the list, displaying each one and attempting to match. The tool would display matching directory and billing information.⁸ Other choices included marking the mailbox as obsolete, or just tossing it to the end of the list. Eventually, all Octel mailboxes got an owner or marked as obsolete.

The matching tool let us work with preliminary data (an earlier snapshot) and one week before the switchover, we froze the Octel database – we had campus announcements out, and after the freeze date, we would process no more changes to the Octel system. We then took that snapshot and finished matching mailboxes with people. At that point, we were able to pre-create all of the new mailboxes on the Unity system. Up to that point, we just had a few pilot users, mostly accessing it via the few VOIP phones we had deployed at that time. Another tool we wrote allowed individuals to pick up their voicemail PIN via the web. We had previously done this for students, and now we were able to expand this for all users.

The cut-over was scheduled for a weekend at the start of the summer (June 6th) – we swung the wires, moved the direct access numbers (there was a number for folks to call to get their old messages) and we were done. On Monday, everyone with a mailbox had to set it up, record their name, set up their greetings, change their PIN, and remember to call the special access number to get any last messages from the Octel. Messages were not transferred from the old system. After 60 days, the old system was turned off and removed from service. If you had not picked up your old voicemail, too bad. We also made the decision to NOT offer voicemail service for students during the summer. This gave us time to handle problems with the fac/staff mailboxes.

Preparing to Merge the Domains

After a year of running a stand alone voice mail system, the decision was finally made to move the

⁸Although we do not charge for voicemail, we often would have owner information for a number (telephone) that we did bill for in the telephone billing system. Sadly, this information did not include a validated owner.

voicemail domain into the regular email domain. This was required to allow us to offer fully unified voice messaging, and would also reduce the support overhead coming from two exchange domains. Shortly after our initial voicemail roll-out, we started a roll out of VOIP service to some new buildings. Part of this roll-out included our own control interface to the Cisco Call Manager, similar to what we did for the Unity project. Unlike the Unity interface, this one used XML [4] and was bidirectional. One of the side effects of this, was the appearance of VOIP phones on the desks of staff who were involved with the Unity project. This was mostly to enable testing of the VOIP service, but proved to be handy in other ways. One of the very valuable features, is that unlike the Intecom PBX, the Call Manager (the “brains” of the VOIP system) understood how to talk to more than one voicemail system, and could talk to them via TCP/IP.

When we first deployed Unity in the Stand Alone configuration, we contracted with IBM to provide us with a consultant to assist us in the planning (including system requirements), configuration and deployment. Ben worked out very well, he certainly knew his stuff and got us going. As an added bonus, Ben was a Rensselaer graduate, so he was familiar with the campus and the general operation. But going forward, we wanted to break the dependency. It got annoying to the technical staff, that when some issue came up, the first reaction was to “Call Ben,” before even attempting to handle the problem ourselves. The decision was made to handle the migration entirely in house; we had to develop our own expertise.

This is Only a Test

We decided that we should replace the Unity server as part of the upgrade. One of the challenges we had with the initial deployment, is that once we went live, we had no test system – all changes had to be made on the production system. It also meant that we had no backup system available. Although the Unity server was a pretty standard box running Windows 2000, it did have some special interface cards in it to talk to the PBX. In the 12 months from the original deployment, the cost of the server hardware came down considerably and since it was going into an existing domain, we did not need domain controllers, an exchange server, etc. So although the new server was part of the production windows domain, it was not in production and we could use it for development and practicing the migration.

When we originally installed Unity, we knew that we would eventually be merging into the win.rpi.edu domain. But since we were concentrating on the move from Octel to Unity, this sort of slipped our mind. One of our first projects to get ready for the merge, was to modify every package, interface, queue and tool to understand the concept of more than one domain. Although the changes were generally pretty

small, it touched everything we had developed. Although a Domain_Id field was added to just about every table, it really only mattered for the call handlers, subscribers and extension (directory number) tables. All other objects were associated with call handler or subscriber and could inherit the domain where needed. The main Unity interface tool had a switch added, to let the user select which domain they were working in. All searches, displays and other functions would be limited to records from that domain. Up until the cut-over, this switch defaulted to the stand alone Unity installation. The other key place for this information was in the command processing stream. Although the Meta_Change_Queue understood multiple queues, the Unity_Maint package had to select the appropriate queue for each command. During the testing of these changes, it was amusing to see a “Create Call Handler” going to the new system, and all of the caller input configuration go to the old system.

We got the new Unity server installed, and made it part of our production Windows domain (win.rpi.edu). Despite having a queue runner program working on the stand alone system, we wanted to merge the queue runner for the Unified⁹ system with the original ADSI_Sync program that we were already running to maintain the windows domain. We also needed to be very cautious in making changes in this domain – just about all of our administrative staff was already using this exchange server and active directory, and disruptions or bad entries were not acceptable. Up until we got very close to switch over, the queue runner process was being run on demand, often times with break points in it, so we could be very sure of what was happening. Another significant operational change, is that although the front office telecom staff got admin accounts on the new Unity server, these accounts were read only. Their only create/modify access was via the interfaces we developed. Fortunately, running the stand alone system gave us time to make available most, if not all, of the needed commands and functions.

Code Generator

Our command interface was much like a checkbook. We made transactions and recorded them, and so maintained a picture of what we thought was the state of the system. But we never had the facility to “balance” the checkbook, and it was possible to miss transactions. In addition, one of our objectives with the cut over to the Unified system, that no one would know that we had done it – greetings needed to be preserved, PINs stay the same, and so on. These changes were all made directly in Unity via telephones, often by the end users and our system never saw them.

After our initial cut over from Octel to the Stand Alone Unity system, our next big project was a Voice

⁹We eventually settled on the name “Unified” for the new system and “Stand Alone” for the old one.

over IP deployment. Despite there being a GUI tool from Cisco, we wanted to have our own interface the the Call Manager¹⁰ to handle provisioning of telephones with better integration with our billing and network systems. The call manager used a SQLServer database internally and provided an XML based interface that could both make configuration changes and return information from the database. It also provided a general SQL interface so that we could make arbitrary queries. Implementing our front end required that we maintain shadow copies of many validation tables in our Oracle database. Writing this interface code was tedious and prone to error, but was also very consistent in format. We developed a code generator; we would define the Oracle shadow table, and then point the code generator at it, and it would spit back paragraphs of PL/SQL code and record definitions that we could then use.

When we returned to the Unity project, we also wanted to get data out of SQLServer, and we had a very limited interface¹¹ that would allow us to query the SQLServer database on the Unity system. We wrote a PL/SQL package for Oracle, that would generate a query to be passed to SQLServer, that would concatenate multiple columns into one, doing appropriate type conversions to produce a CSV format return that we could then write more PL/SQL code to parse and store. Getting all of this to actually work (part of one query required seven single quotes in row to get the appropriate single quote in the final result) cried out for automation, and so we wrote another tool to handle all of this. One of the things that SQLServer has, is a set of tables known as the data dictionary, that define all of the table and columns (and other things) in the database. With this, we could query the data dictionary and get back descriptions of all the table and columns in the database and store them in the Oracle database. We then added a tool that allows you to select a table, and columns of interest, the type of query (just one record based on a primary key, a list of all keys in a table, etc.) and eventually build up a set of queries of interest. Then with this, we would generate an entire PL/SQL package with record definitions, parsing routines, query routines, etc. This package could then be compiled in Oracle in used by other parts of the system.

The first application of the code generator, was to regenerate the interface used by the code generator itself to get data dictionary information. The code generator made it very easy to access almost all of the data we needed from the SQLServer database on the Unity system. We could get lots of the mailbox configuration information, some statistics about the mailboxes and even the PIN. We actually could only get an MD5 hash of the PIN, but we discovered that when

you pass a 32 character hex hash of a PIN to the `Set_DTMF_Access` routine, it would recognise it as already hashed and set it directly. Thus although we could not determine what someone had set the PIN to,¹² we could determine that they HAD set it, and we could copy that value to the new system.

Call Manager Integration

Between our initial deployment and the migration to Unified messaging, we had one major change in our environment; the VOIP deployment. When a voice mailbox is configured, you need to indicate which phone switch it is on. This is so Unity knows which switch receives the MWI. As people were moved from the PBX to VOIP, we sometimes forgot to update Unity, so a few days after someone got moved to VOIP, we would get a trouble call that their message waiting light wasn't working anymore. We modified our call manager configuration tool, to check the Unity configuration and change it automatically if needed. This greatly reduced the number of MWI trouble calls we were getting. This type of cross system integration is a nice feature of doing our own tool development.

Recorded Names and Greetings

Much of the subscriber configuration information was stored in the SQLServer database, and the actual voice messages were stored as email on the Exchange server. But the recorded names and greetings were stored as .wav files on the file system on the Unity server. We could at least get the name of the file from the SQLServer database. Our next step was to define a new command for the queue runner that would ask it to open a sound file and save it into the Oracle database. The file would get converted to Hex, passed to a an interface routine in Oracle, that would convert it back into binary and store it as a BLOB.¹³ We were already saving ID card photos as BLOBs and we had reasonable support for managing this information and spitting it back out via web tools. This was pretty neat, as it allowed our admin tools to "play" the greetings and recorded names for subscribers and call handlers. We were able to "recognize" names that we had recorded and greetings that we had generated by the file names, so we didn't bother to actually save these in the Oracle database. Collecting names and greetings was not a fast operation.

Since it took so long to copy all of the recorded names and greetings, we next wanted to develop a system that would just get new recordings. Unfortunately, although the SQLServer database had the file name, it did not know the file size or last changed information. So, we added yet another command, a request for file information. When the queue runner got this, it would get the file size and last data info, and store that in the

¹⁰The Call Manager is the "brains" of the VOIP system.

¹¹Only the first column of the query would be returned and the total returned data was truncated at 32 Kbytes.

¹²OK, 5-8 numeric digits and known hash, we could brute force it.

¹³Binary Large Object – how Oracle can store large binary data values such as a .WAV file.

Oracle database. This made for some complex update code; it would check to see if a subscriber had a recorded name. If it did, it would send a request for the file info for it. But it might be minutes or even hours before it got a response (an update in the table), so it would request info on a lot of files, and mark them all as pending. Some time later, it would look at the pending to see if they had been updated, and if so, it would then ask for a fresh copy of the file. A more direct query approach have made writing this a lot easier. But in the end, we had a system that could, in the span of less than a day, ensure that we the most recent greetings and names.

Migration Tool

When we moved from Octel to the Stand Alone Unity system, we had written a migration tool to help us with that project. So, we wrote a new tool that would let us examine all of the call handlers and subscribers on the stand alone system and figure out how to map them to the Unified system. This also provided a platform to test the migration procedures. When we had installed the Unified system, we did not populate it with subscribers, and just set up a few call handlers and templates to provide a starting point.

The first folks migrated were part of the pilot group – generally the technical people who were actually doing the migration, and would not (or could not) complain if their voicemail got messed up. Since these were going to be “unified” clients, we did not need to change their mailbox name. The first step in migrating someone, made an entry in the migration table, and changed their status. The first state was generally “wave wait” – which also generated a request for the sound files. Once the sound files arrived, the new name could be set and when we were ready, we pressed the “create” button. As time went on, we got better, identifying more special cases we had to handle, and getting all of the configuration information that we needed. These early adopters did not get their messages copied over until much later.

The tool was then changed to list sets of subscribers, based on class of service, who were not yet in the migration table. Check boxes made it pretty simple to decide if we wanted to migrate them, and what their new mailbox name should be. This allowed us to review every subscriber and call handler well ahead of time and get them started down the path. Once a subscriber or call handler was created, a link to the new entry was also stored in the migration table, so we built up a record of everyone and everything that got moved.

Prior to Cut-over Weekend

We had originally planned on doing the migration during the week between Christmas and New Years. This had the advantage of the students being out of the dorms and most of the staff on vacation. However, the wife of the exchange administrator was

due to deliver their third child¹⁴ that week, and since it wasn't critical that we migrate, we decided to wait until the entire upgrade team was available. The cut over was moved to the first weekend of Spring break. The real driving force was that was the first weekend that all members of the upgrade team (at least the key members) did not have other obligations. It certainly is nice to be able to select a time that works well for everyone.

Despite having had the pilot group using the new server, and many of them still unified, there were still things that we were unable to test. For example, we were pretty sure that creating 5000 new mailboxes on our exchange server would not have any unexpected side effects, but we wouldn't know until we tried. So two weeks before the cut-over, we started creating the new mailboxes using the migration tool. After the first few batches of 100 went through ok, and backup ran ok, we sent over 1000, things again looked ok, and we were able to finish the bulk of them a week before the cut over. We also added some bulk processing options to the migration tool, to speed the processing along.

How Long?

Another process we started early, was setting up the sound files. We did not want to load the sound files too early – users of the voicemail system were still updating their greetings and we did not want to “freeze” changes until shortly before the cut over. But since our collection process was able to detect files that had changed, we started loading sound files, and a good thing we did! Although everything worked in tests, and the extraction of all sound files only took 36 hours, moving the sound files from the database and converting them back into .WAV files on the Unified system was very slow and our initial estimate was almost two weeks! Some research into that determined that most of the time was in the hex to bin conversion, and some web searches revealed discussion of just how slow some of the library routines that did this are. An alternate library gave us a speed increase of 20, which was good enough to get us in under the deadline.

We wrote another program that would copy mailboxes from the Stand Alone exchange server and deposit their contents into the Unified exchange server. It would take a crosswalk file, since almost all of the mailbox names were changing as part of the process. This also was going to append email, since members of the pilot group already had exchange mailboxes as their primary mail service. We couldn't figure a good way to test this – although we had a Exchange test domain, it is isolated from the “real world,” since it tries very hard to get out. We were also concerned that any testing here would raise more questions than it answered. Since this program was based on some tools we had used during an Exchange recovery operation, we had some feel for the timing,

¹⁴Katia Birgit Hill, 7 lb, 13 oz

and we were confident that it could run in less than two days – well within our planned upgrade window.

Cut Over!

Friday at 6:00 PM rolled around, and we started going through the step by step upgrade plan. We had taken some concepts from Brent Chapman's Invited Talk [1] from LISA last year, and we had a communications person and a communications plan, so that people who needed to know what was happening, were informed as milestones were reached (or not).

In the months leading up to the cut over, we had developed an implementation plan including time estimates, deadlines, dependencies and staff for each step. We also developed a more detailed tactical plan for the upgrade weekend, including communication schedules (i.e., – we were going to give the CIO a status report at 9:45 PM, he wasn't supposed to call us prior to that time.) The plan even included dinner menu selections for the team members. Some aspects of the plan called for processes to run – instead of standing around watching, most of the team could eat dinner.

We also had a test plan, which involved other people from the division testing aspects later on in the weekend, as well as specific things to test. We also requested that these testers try those same tests on the existing system prior to the cut-over, so that they would know what to expect. We did not want to waste time troubleshooting a problem that existing on the old system and was “successfully” migrated (and we found bunches of these.)

Final Results

We had reserved the entire weekend for the upgrade. Within an hour of the start, the wiring changes were done and the new system, was accepting messages for people on the PBX. Within an hour after that, we had moved most of the VOIP people over. We encountered some problems that did not show up in testing, where some phones did not want to change their voicemail profile and continued to direct calls to the old system. Migration of old messages was scheduled to start Saturday at 9:00 am and by noon, we had copied all of the voice messages over from the old system to the new system, and although we were still doing testing, for all intents and purposes, the upgrade was done and most of our users did not notice any outages. Their messages were available, their greetings and recorded names were intact and their PINs still worked.

Oops

We missed a few things. Although we were finally able to get all of the VOIP phones switched over, we overlooked the tool used to add new phones – it was carefully assigning new phones to the old system. Fortunately, once we figured this out, we were able to correct it and had the users get their messages from the old system. A few months after the cut-over, we were still occasionally finding new VOIP phones

with voicemail on the old system. This was finally tracked down one module that had a literal string with the PKID of the old voicemail profile embedded in it. This was noticed when the old voicemail profile was deleted from the VOIP Call Manager and certain configuration functions started to fail.

We had instructed our users that changes made after a certain date (a week before the cut-over) may not be migrated. We had a few cases where people recorded alternate greetings and enabled them and left for vacation. Unfortunately, the process to migrate greetings was not synchronized with the process that copied the state of the greeting switch, so as a result, a few people had voicemail greetings wishing folks a “Happy Thanksgiving” during the week of spring break.

The Unity system (at the release we had), could be configured to work with two phone systems. So in the initial deployment, the IBX was switch 0 and the call manager was switch 1. When the new Unity system was installed, the first switch that was configured was the call manager so it became switch 0, and the IBX was switch 1. This didn't matter too much, since phones are generally configured by name. What this did do, was change the default switch for lots of mailboxes. Since this didn't matter for mailboxes that didn't have a real phone, this shouldn't be a problem.

One of the nightly jobs the Unity server runs, is to refresh the Message Waiting Indicator (MWI) on all the telephones. This is that nice little red light that tells you that you have a message waiting. Or for the folks with standard telephone sets, the “stutter” dial tone, so that when you pick up the phone, you get some indication that you have voicemail. Of course, we have found that many of the folks who have the standard phones, don't make a lot of phone calls, and so never pick up the phone to hear dial tone, and so never learn that they have voicemail. This also doesn't work well where you have shared phones. Which mailbox controls the single light? One option here is to use email notification – when you have a new voicemail, it sends you email telling you about this. While this makes no sense for unified people (sending you an email to tell you that you have an email?), it is nice for folks with just the basic voicemail service. Many of our mailboxes are set up with 5 or 6 digit “extensions” (fac/staff shared phones that were migrated from the Octel system often kept a 5 digit extension, where the first 4 digits matched the phone, and the 5th was the “press N” value.) So every night, Unity would attempt to update the MWI setting for 5000 phones – many of which did not actually exist. The connection to the IBX was a serial line – so this update would take almost an hour. While this was going on, new MWI changes would get stuck at the back of the queue. This wasn't a big problem at 4 am, but was mildly annoying.

When the new system went live (or actually when created 4900 more mailboxes), we noticed that the default switch was the call manager and not the IBX, but hey, the update of the MWI would run a LOT faster via TCP/IP. Shortly after that, we noticed that the job that was processing the call detail records (CDR) for phone billing, was timing out – it was taking too long to run. Further investigation revealed that each of these MWI updates was generating two CDR records for each mailbox, one for invalid extension and one for service not available. This increased the number of CDR records each day from around 2000 to over 10,000. This was also filling up the CDR space on the call manager a lot faster, and wasting space on the billing system. For a short term fix, we wrote yet another process to pre-delete these CDR from the call manager before the billing job runs. Who would have thought that such a trivial decision would prove to be so annoying.

Another thing that bit us was with licensing limits. We originally had 5000 licenses, and our usage was getting really close to the line, so we got 50 more. This involves loading a new license file and restarting the system. We did this and operations continued. However, this happened after we had built the new system, but before we cut over, so it was still running with the license limit of 5000 and not 5050. Shortly after the migration, we learned that you do NOT want to exceed your license limit. An updated license file, a brief outage and full service was restored.

Conclusions

In retrospect, doing the initial installation as a stand alone exchange domain, and then later migrating it into our production email domain was a good thing. We had no operational experience with the Cisco Unity product, and despite having run the previous voicemail system for 12 years, there were some significant differences. We had to make a lot of decisions when we first deployed Unity, and several turned out to be sub optimal.¹⁵ We were able to correct a number of these as part of the migration process.

We also got a test system out of the whole process. One of the first things we want to try, is does our system restore actually work.

Are We Done Yet

A rather annoying oversight from an implementors standpoint, is that we had not defined a way to determine when the migration was “done.” There is a different mindset, sense of urgency, etc. between “Cut over Weekend” and “Normal Operations.” It may seem like a trivial point, but it has an impact on the people involved. One of the problems we had, is that there were things that were configured wrong on the old system and we “successfully” duplicated those wrong configurations on the new system. It was nice

¹⁵Which is a nice way of saying “wrong.” But we didn’t know any better at the time.

to finally just fix problems as they were encountered, and not worry was it a migration problem or a new or even an existing problem. It was really annoying to get a complaint about someone’s message waiting indicator not working, and after an hour of digging, determine that it NEVER worked in the first place.

We also left the old system on line – perhaps for too long. This allowed some folks to continue to use the old system without anyone knowing it – which led to some confusing problems. This was only a problem with the call manager. The PBX was a hard switchover, a physical cable move, but the call manager continued to talk to both system, and it wasn’t until we deleted the voice mail profile on the call manager the referenced the old system, did we shake out the that last few references to it.

Culture Clash

The email folks and the voicemail folks had developed many years of policies and procedures. With the merging of the two worlds, these differences need to be identified and resolved. For example, in the past when a new person got a telephone, they would often have to flush out the messages and greetings from the previous owner of the phone. We have NEVER re-used email accounts! What was once a reasonable and acceptable practice, is now obsolete!

Another issue we have the actual deletion of mailboxes. In the stand alone domain, the module that deleted a subscriber record, would also delete the exchange mailbox. But with our “real” windows domain, the administrators still review and delete mailboxes by hand. I expect once the mailboxes from last springs students hits the “delete” queue, they may reconsider this. For now, we delete them as subscribers (which frees up the Unity license) and they are marked as disabled in Exchange.

One of the more annoying problems, is dealing with full mailboxes. With the stand alone domain, we had a rule that would permanently delete messages that had been “deleted” by the user. Mailboxes would still occasionally fill up, but that didn’t really cause much of fuss. Unfortunately, we have some users of the exchange service, who have decided that the trash can, is just another folder and that it is a good place for long term storage (I would hate to see their office). We have missed our chance to put the same rule we had on the stand alone system, trash gets emptied automatically!

Futures

Although we are now positioned to provide unified voice messaging to anyone on campus, there are some issues to be worked out. We have the basic problem that while anyone with a telephone (and in some cases, that may not be a requirement) can have a voicemail box, the requirements to get an exchange mailbox are much stricter. There are also some additional features we would like to offer, and some other minor problems that we would like to fix.

Partially Unified Environment

Unlike many Unity installations, we will be operating in a mixed environment where many people are not unified. This can result in some potential issues for our non unified users. Although we have attempted to hide voicemail only users in active directory, and we also tag their names with the word “voicemail,” these addresses can still “leak” out. The simplest way would be when a unified user receives voicemail from a non unified, but on campus, user. The voicemail will be from a recognizable name (i.e., “David Hudson (Voicemail).” It is a trivial matter in outlook to then add that address to your address book, and then you can easily access what is supposed to be a hidden name. Ok, so what is the problem here? Say some wacko (perhaps with tenure) leaves me voicemail saying that he wants some specific hostname changed. As a unified user, I forward this, after typing some comments like “Ignore this idiot – let me know if he contacts you,” but I accidentally get Dave’s voicemail entry. Dave then listens to his voicemail, is told by Unity that I forwarded a voicemail to him, and plays him the original message. But since there is a voice message, it does NOT read him my comments. If I had just sent Dave a regular email to his voicemail address, he would have been told he has an email (which might confuse him), and it would then read it to him, but in this case, since there was voice, only that would be played.

There are a few other oddball cases like this, but we were concerned enough that we have delayed widespread deployment until we can resolve this. Work is underway to add some hooks in the Exchange server that will detect these cases and generate a bounce to the sender. Once that is done, I expect that we will resume the more general deployment.

Student Voicemail Becomes Opt-In

One of the things we had overlooked in this project, was anti-virus scanning. In the original stand alone deployment, we did not need anti virus scanning, as the exchange server was pretty well isolated from the rest of the world. However, in the unified environment, we need the scanning, and you have to scan all mailboxes. This is licensed on a per mailbox basis, so there was a jump in the licensing cost. Cisco is also changing their licensing, and the per mailbox cost is going to be going up.

Historically, we had provided voicemail service to all on campus students. Not only was the old Octel system paid for, the vendor had gone out of business, so even if there had been per mailbox licensing costs, there was no one to pay them to. It was easier to create mailboxes for everyone, and ignore the idle ones. Now that model has changed, and there is significant costs for mailboxes. We don’t mind paying for active ones, but paying for idle ones is a bit annoying. Many of our students come to campus now with their own cell phones. One of the things we have noticed, is a large

drop in long distance revenue from students – they are no longer using their dorm room phones to make long distance calls. What is more, they don’t seem to be using them all that much to receive calls – at least many are not bothering to set up their voicemail boxes. We did some analysis of usage; one of the fields saved by Unity is the last time a mailbox was opened via the phone. For the spring semester, of the 2500 student voicemail boxes we had on line, only 500 actually called it! We had enabled email notification for students, and all mailboxes had a recorded name (done automatically), but it was pretty clear that we were wasting a lot of licenses.

After some discussions with the CIO, Residence Life and the Office of the First Year Experience, we decided to make student voicemail an opt in service. The web tool they use to pick up their voicemail PIN, will let them request/create a voicemail box if they so chose. After about three weeks since the dorms opened, we have had 370 out of a possible 2925 students request a voice mailbox for their dorm room.

Department Administrator Tools

We have identified departmental administrators for each department. These people can update directory information for people in their departments, as well as issue short term VPN/Wireless access accounts and manage guests. We will be adding a tool to help these people administer the voice mailboxes owned by their department or people in their department. This will likely be rolled out when a similar tool to administer VOIP equipment is released.

User Call Handler Options

One feature we hope to release to users once we understand the billing issues, is the ability to maintain your own caller input selections. For instance, when you listen to my message, you are given the option to press six, and your call will be transferred to my cell phone. This allows people to contact me via my cell phone, yet I don’t need to give out my cell phone number.

There are a number of other advanced features in the Unity systems that we may be exploring as well.

References and Availability

The bulk of this project is written in PL/SQL. The core package is the Unity_Maint package (see Figure 2) that provides access the to Oracle tables that store our version of the configuration, and communicates with the Unity server via the Meta_Change_Queue package. This package is basically site independent – none of our business rules are applied here, just the basic communication and configuration. The bulk of the site specific operations, and those requiring work in more than one table (such as creating ECPs) is in the Unity_Functions package. Later efforts to work in expiring mailboxes was put in the Unity_Expire package. This division was mostly to accommodate a lot of changes in expiration processing without impacting tools using the Unity_Functions package. The primary

administrative tool is provided via the Web_Unity_Admin package. The student voicemail tool predated this project by several years and is in the Web_Voice-mail_Admin package which in turn used the Tcom_Voice-mail_Maint package. These existing packages were joined to the new project via the Unity_Dorm_Functions package. Some of this structure is the result of when things were deployed and several packages could be merged.

The Unity Queue Runner program took commands from the Meta_Change_Queue, translate them via the Unity_Record package and then processes them appropriately. Although in our original Windows 2000 integration process we were maintaining Active Directory directly, in this case, we made the changes in the Unity server (which has it's own SQLServer database), and the Unity application would make the changes in Active Directory. The Unity application would also create Exchange mailboxes as needed (Deleting mailboxes has been disabled for now).

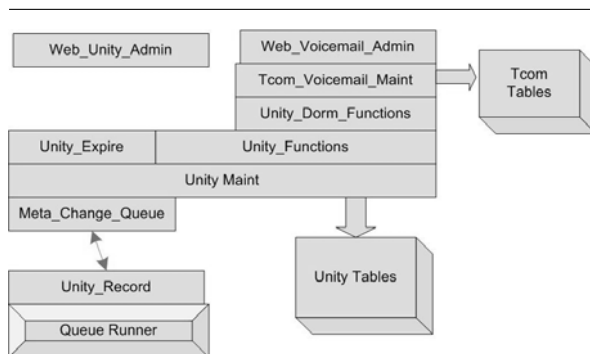


Figure 2: Code Structure.

Most of the PL/SQL source code for the Simon system as well as the full table and view descriptions are available via the web at <http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html> and <http://www.rpi.edu/campus/rpi/simon/misc/Tables/SIS-index.html>. Some of the modules that interface with the Cisco Call Manager are currently under contract with another vendor, and we are not making them publicly available until the details of the contract are worked out. The migration tools were changing during the course of the migration, what we used at the start is different from tool we used at the end of the project. These tools were not intended for extended use, although we will keep the source code around in case we migrate again. There are also a number of scripts that were put together as needed.

If you ask nicely, I will try to answer questions and might be able to dig out some of the C and JAVA code that makes up other parts of the system.

Acknowledgments

I would like to thank Rudi van Drunen for his shepherding of this paper with me. I also want to thank Rob Kolstad for his excellent (as usual) job of typesetting this paper. I also want to thank the other

members of the Unified Messaging Team here at Rensselaer for their contributions to the project, as well as my girlfriend Joyce who encouraged me throughout the entire process.

Author Biography

Jon Finke graduated from Rensselaer in 1983 with a BS-ECSE. After stints doing communications programming for PCs and later general networking development on the mainframe, he then inherited the Simon project, which has been his primary focus for the past 15 years. He is currently a Senior Systems Programmer in the Communication and Middleware Technology department at Rensselaer, where he is developing identity management systems and integrating Simon with the rest of the Institute information systems. In addition to the Simon project, Jon is also involved with the support of the Telecommunications billing system, and providing data and interfaces for Unity Voice Messaging and CISCO VOIP deployment projects at Rensselaer. He hopes to someday find out what Middleware is. When not playing with computers, you can often find him merging a pair of adjacent row houses into one, or inventing new methods of double entry accounting as treasurer for Habitat for Humanity of Rensselaer County. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via <http://www.rpi.edu/~finkej>.

Bibliography

- [1] Chapman, Brent, "Incident command for IT: What we can learn from the fire department," *The 19th Large Installation Systems Administration Conference (LISA 2005)*, December, 2005.
- [2] Finke, Jon, "Institute white pages as a system administration problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp. 233-240, USENIX, October, 1996.
- [3] Finke, Jon, "Embracing and extending Windows 2000," *The Sixteenth Systems Administration Conference (LISA 2002)*, USENIX, November, 2002.
- [4] Finke, Jon, "Generating configuration files: The directors cut," *The Seventeenth Systems Administration Conference (LISA 2003)*, USENIX, October, 2003.
- [5] Finke, Jon, "Meta change queue: Tracking changes to people, places and things," *The Eighteenth Large Installation Systems Administration Conference (LISA 2004)*, pp. 231-239, USENIX, November, 2004.
- [6] Finke, Jon, "When worlds collide: The two-sided sword of technology integration," *Login: The USENIX Magazine*, Vol. 30, Num. 3, pp. 6-7, June, 2005.
- [7] Microsoft, *Visual c#.net*, 2001, <http://msdn.microsoft.com/vcsharp>.

Fighting Institutional Memory Loss: The Trackle Integrated Issue and Solution Tracking System

*Daniel S. Crosta and Matthew J. Singleton – Swarthmore College Computer Society
Benjamin A. Kuperman – Swarthmore College*

ABSTRACT

For part-time sysadmins, a record of past actions is an invaluable tool that provides guidance in repairing or extending system services. However, requiring sysadmins to keep a detailed log of changes made to a live system can often seem like a low priority task when compared to addressing long and growing to-do lists. This problem is worse if the system administrator is a part-time volunteer and an overworked student. In this paper we present Trackle, an integrated trouble ticket and solution tracking system which takes the legwork out of creating and maintaining this sort of institutional memory. Furthermore, Trackle is designed to allow untrained student sysadmins to bootstrap their knowledge by peeking over the shoulders of their more experienced colleagues – even if those colleagues graduated years earlier. We accomplish this by tracking the exact actions taken by sysadmins, showing what lines were changed and in which configuration files. We allow experienced and inexperienced sysadmins alike to freely annotate and cross-reference these shell session logs through an integrated Wiki web interface.

Introduction

The Swarthmore College Computer Society (SCCS) is a group of student volunteers who provide services to more than 2,000 current students, alumni, faculty, and staff of Swarthmore College. We provide UNIX shell accounts, email, group mailing lists, web space for individuals and student groups, Wikis, database access, and general computer expertise. We also maintain a public lab of eight Debian GNU/Linux and two Apple Mac OS X computers. The sysadmins meet weekly for an hour to keep up to date on projects, problems, and planned improvements, and communicate via email between meetings.

Unlike many large UNIX administration setups, the SCCS has no full-time trained staff. The student sysadmins are hired once a year, usually in groups of two or three, and most graduate after serving as a sysadmin for two years. At any time we have about eight sysadmins on staff. Since a part of SCCS's mission is to provide an environment in which interested students may learn the art of system administration, we recruit students with technical backgrounds ranging from "I can email and browse the web," to "my desktop PC is a Beowulf cluster." This disparity in background and the high turnover rate presents us with the dual problems of how to preserve our collective knowledge and how to train future generations.

For many years, our primary means of training sysadmins has been the SCCS staff email list and archived diaries. The diaries are a cluttered repository; we use the staff email list as often to tease one another as to discuss pertinent policy, administrative issues, or

help requests from our users. Though sysadmins are supposed to report changes to our servers and lab machines to the list, we often have to rediscover particular configurations when problems arise. Furthermore, mbox mailboxes are a hard format to browse, particularly for new, inexperienced sysadmins who are eager to learn but more comfortable browsing the web than navigating the sometimes murky waters of UNIX shell prompts, commands, and filesystems.

In the last three years, SCCS staff have used a private Wiki to document changes, configurations, and internal policies as well as for collaborative planning. We have found the unconstrained and highly inter-linked environment very valuable to our operations, but relatively little of the content on the Wiki is pertinent system configurations or records of changes made. As with the email diaries, the extra time and effort required to self-report changes made to our systems presents too high a hurdle for our volunteer sysadmins, each of whom has a range of other academic and extra-curricular commitments.

For training, we needed a system that preserved the ability to associate high level conceptual descriptions of desired configurations or problems with the particular shell actions or configuration file changes to put those ideas into effect. Unlike the Wiki and email list, we wanted a system that did not require sysadmins to take on the extra burden of having to write up complicated records of their actions after long, sometimes frustrating problem-solving sessions. Furthermore, since our sysadmins are hired without any requirement of previous experience, we wanted a tool which could help new sysadmins learn the subtleties

of UNIX administration and our particular setup without requiring any former familiarity with UNIX.

To respond to these concerns, we created Trackle, a web- and console-based integrated issue and solution tracking system. Each problem reported to the system, either by our users or by SCCS staff, creates a ticket in the issue database. Open issues may be viewed through the web where they can be freely annotated, edited, and cross-referenced with the built-in Wiki; or through a console interface, from which a tracked shell session may be started. All actions taken in the shell session, and all files changed, are recorded in the Trackle database, and create an entry in the ticket history log. These shell session logs are then editable in Wiki fashion.

Existing and Related Work

We first attempted to discover if any existing tools could be used to solve this problem. Since a sysadmin's job is primarily task-oriented, we began our search with trouble ticket and issue tracking systems. Issue tracking systems seem to have lost appeal in the literature since the late 1990s. Before that, there were one or two issue tracking papers per year in LISA, most of which explored different extensions to the basic trouble ticket idea.

One early tool, Request 2 [8], was designed with a pedagogy of training junior system administrators in mind. With Request 2, senior sysadmins could divide and delegate work to less experienced admins, and oversee their work or offer advice.

PLOD, the Personal LOGging Device [6], enables sysadmins to easily self-report by providing a set of command-line Perl scripts which the sysadmin may use to make notes to him or herself about work just completed. PLOD simplifies the tasks of keeping records consistently and making them centrally available. PLOD did not include, and was not integrated with, any issue tracking system.

The LOS Task Request System [9], designed several years ago by another SCCS sysadmin, allows users to create ticket-like task requests through a web interface. Each task request is associated with a task description, which includes validation logic and the commands required to execute the task. This alleviates sysadmin workload by automating the tedious tasks of collecting and verifying information for repetitive changes to the system.

A related field of more current interest is configuration management and version control. Configuration management systems attempt to centralize or abstract configuration details for large numbers of similar hosts. Tasks include initial setup, maintenance, troubleshooting, and incorporating local changes back into the global description. For a more careful consideration of the history and future of configuration management, see [1, 3, 7].

The SCCS currently has no plans to use a configuration management system at our site. Our two primary servers do not share enough common configuration to yield much advantage from such a system, while our public lab machines, which are configured similarly, are periodically reinstalled to a known consistent state for security reasons. For these reasons, we do not believe we would gain enough benefit from a configuration management system to justify the setup effort.

Much of the time our sysadmins spend working involves user-owned files, such as spam filter settings, dotfiles, and web access configurations. Since these files may be created and deleted by users at any time, a configuration management system would not be a good choice for monitoring changes to these files.

Moreover, though configuration management may be considered an industry best practice, we believe no automatic configuration system can adequately replace a sysadmin's ability to get dirty with manual configuration and hand-tuning of UNIX services and applications. We would rather our sysadmins learn how to discover a problem at its source and implement a solution appropriately than merely learn how to control a single piece of software, no matter how powerful it may be.

One of the chief goals of Trackle is to provide a platform for training inexperienced sysadmins. We believe that it is important to gain an understanding of UNIX cause and effect by observing and learning about configuration files, logs, commands, and scripts. If new sysadmins only learn to rely on abstract configuration management systems, they might find themselves unprepared to deal with problems outside the scope of those systems.

None of the systems we considered deploying covered our needs for a system to track issues, actions, and train new sysadmins. A ticket tracking system would help organize and abstract the institutional memory knowledgebase, but would still require self-reporting. A configuration management system would keep accurate records for the files that it tracked, but would not provide a good platform for training incoming sysadmins. Having searched and failed to find just what we wanted gave us a chance to reevaluate our problems, and we decided that only by implementing a custom solution could we meet our requirements.

Design of Trackle

From the beginning, the design of Trackle has been motivated by a philosophy of minimalism and simplicity. We believe that the most useful system for our needs is one that imposes as little as possible on our sysadmins in terms of new workflows and interfaces to learn. We want Trackle to stay out of the way of sysadmins, but at the same time to automate as much of its own data collection and presentation tasks as possible.

It is always difficult to find the right balance of the automatic and the manual, the consistent and the customizable. Given the sort of information Trackle manages, we believe the right place for the cut is at data collection and initial presentation. Creating the complex knowledge structures through annotation and cross referencing is left to the sysadmins as a secondary task, since it is not one that a computer is likely to do well.

Trackle is intended for smaller groups of sysadmins with a relatively low volume of requests. It is designed to be used effectively by both expert and inexperienced sysadmins. Trackle's information organization methods are best suited to small groups who have time to annotate and cross-reference collected data. The goal of allowing this rehashing and reorganization of information is to provide a platform for self-directed training of inexperienced sysadmins on their own time.

Requirements

Two classes of users: Since the SCCS naturally has two classes of users, sysadmins and SCCS end users, Trackle was designed to accommodate both types. Differentiating the web interface for the two types of users allows us to improve both security and convenience. Unauthenticated end users are denied sensitive information, such as email addresses in tickets or files in logged shell sessions, and are not shown potentially confusing prompts when creating tickets. Sysadmins, on the other hand, need access to all the information about tickets and shell logs, and should know enough to understand the more detailed ticket attributes.

Low barriers to use: One problem with existing ticket tracking systems is the often overwhelming amount of information that is requested to submit a ticket. With Trackle, we wanted to make filing a ticket as easy as possible so that end users and sysadmins alike can move quickly through the web forms. Our interfaces are designed to be intuitive so that they can be used without having to waste time reading documentation.

For end users, we wanted Trackle to be as easy to use the first time as the tenth. End users do not have to register accounts with Trackle to create or subscribe to tickets. Instead we use an email confirmation system to verify an end user's identity.

High-level organizational tools: Trackle's central goal is to provide a framework for flexible representation of the collected data. It is designed to facilitate Wiki-like annotation and cross-referencing rather than imposing a fixed organizational structure. Objects in Trackle support Wiki formatting, enabling them to link to each other. With these tools, the recorded data can be arranged, indexed, and presented in the ways most useful to the sysadmins who need access to it.

Few dependencies: Trackle was designed to be dependent on as few external software packages as possible. It is specifically designed to work with the

packages and versions available in the Debian Sarge GNU/Linux distribution (our deployment platform); however, we wanted to make Trackle freely available for any interested groups for use on many platforms. To ensure easy portability, it is not closely tied to any particular versions or packages.

Ticket System

As discussed earlier, we decided to use a ticket tracking system as the basis for Trackle's higher-level categorization of captured data. We briefly considered RT,¹ but decided that it would be too confusing and cluttered with information to be useful to inexperienced sysadmins. Instead, we decided to use a relative newcomer to the trouble ticket world, Trac,² as our foundation.

Trac is a BSD-licensed bug tracking system with integrated version control and Wiki. Trac embodies a minimalistic approach to ticketing which fit well with our design goals: it presents a clean, interface intuitive to both our end users and sysadmins, and has relatively few features and ticket attributes specific to bug tracking. Many of the features we wanted (easy annotation of tickets and shell histories, high-level organization and cross-referencing through the integrated Wiki, file change visualization) were already implemented, and adding our own functionality was very easy due to the well-organized Trac API.

However, Trac required more than cosmetic changes to fit our needs. Despite its flexible permission system, Trac does not natively support multiple classes of users, nor does it have ticket locking. Additionally, we had to modify the ticket status logic to accommodate unconfirmed tickets.

Tracking File Changes

Tracking file changes was the biggest challenge encountered when designing Trackle. Because we do not use any configuration management or version control, we needed to ensure we could get a before and after view of each file modified during a sysadmin's shell session. We considered several alternative strategies before eventually settling on a custom interposition library. The possibilities may be roughly split into kernel-space solutions (LVM snapshots, UnionFS, FUSE, C2 audit logs) and user-space solutions (plugins/history interpretation and library interposition).

LVM snapshots: The Linux Logical Volume Manager³ (LVM) provides a way to create a time-frozen snapshot of a volume. In Trackle, we could use a LVM Snapshot to capture the pre-shell tracking session state of the system and then compare the two filesystems at the end of the session. LVM works below the filesystem and captures changes by tracking changed disk blocks rather than changed files. To use LVM with Trackle we would have to relate changed

¹<http://www.bestpractical.com/rt/>

²<http://trac.edgewall.com/>

³<http://sourceware.org/lvm2/>

disk blocks to changed files, or compute a recursive diff between the two hierarchies; the former would be prohibitively hard and would be tied to particular filesystem implementations, and the latter would be prohibitively slow. It is also unclear how to differentiate what files were changed by the user during the shell session and which files were merely changed by other users or processes during that time. Additionally, it would impose LVM as a runtime dependency for Trackle and limit Trackle to use on Linux.

UnionFS: UnionFS⁴ is a meta-filesystem that stacks one or more existing, mounted filesystems into one hierarchy. It supports copy-on-write so that a read-only filesystem can be made to appear as a read-write filesystem. This functionality is used by many Live Linux Distributions, like SLAX⁵ and KNOPPIX.⁶ When operating on a file in a UnionFS stack, the top-most instance of the named file is used. This means that in order to keep a initial copy of the file for Trackle's use, the underlying filesystem would have to be mounted read-only. Since it would be impractical to remount the entire filesystem hierarchy read-only, Trackle would have to create a read-only bind mount of the root hierarchy, to use as the bottom of the UnionFS stack. On top of that would be placed a read-write filesystem mounted elsewhere to capture the revised versions of files. Because this third filesystem is hierarchically removed from the active root filesystem, any changes made during the shell tracking session would not be reflected to the root filesystem unless manually copied (for instance, by a special Trackle command) or when synchronized at the end of the session. Additionally, UnionFS requires the insertion of a new kernel module, which some sysadmins may be hesitant to allow. At this time, UnionFS is only available for Linux.

FUSE: The Filesystem in Userspace⁷ (FUSE) technique works similarly to the interposition library approach we finally adopted. By use of a special kernel module, some or all filesystem-related library calls can be delegated to a user space daemon. Trackle could use FUSE to make note of which files the user is accessing and copy the files' initial state before returning from the user's library call. This provides about the same level of flexibility as using an interposition library. Using FUSE requires loading a kernel module and mounting and unmounting filesystems. In order to use FUSE with Trackle, we would have to create additional setuid-root binaries to handle these tasks. FUSE is currently only available for Linux and FreeBSD.

C2-like audit logs: Many operating systems include auditing facilities to track file accesses as one of the requirements for a trusted computer system [10, 2]. There are projects to bring this type of auditing to

GNU/Linux such as SNARE for Linux.⁸ and SELinux⁹ These systems require kernel modifications, and are typically enabled for the entire system instead of just a process and its subprocesses. Consequently, it is difficult to dynamically enable auditing, which results in the generation of overwhelming amounts data. Additionally, these audit logs only notify us that a file has been modified after the fact, limiting our ability to determine what changes were made.

Plugins and shell history interpretation: In theory it is possible to capture the majority of file changes made by a sysadmin by writing plugins for the editors Vim and Emacs to record files opened or saved, and by looking at the command history of the shell to infer which files might have been modified. This approach can never be more than approximate as there are many other ways to change a file other than by these two editors. Even writing plugins for these editors will not capture changes made to underlying files by wrapper programs like vipw that act on temporary files. Additionally, editor plugins do not capture file changes made by users at the command line either by shell redirection ("echo stuff >> outfile") or file-related commands (mv, cp, rm, chmod, etc.).

While parsing a shell history file might be able to discover these kinds of changes, it would have to happen after the commands changing the files had run. This would prevent us from tracking file changes, and the list of command patterns to check would be quite large. Wrapping all file change tools and implementing a custom shell would impose a large (if not insurmountable) maintenance task.

Library interposition: Most binary executables on UNIX are dynamically linked and access files through standard C library functions (open, unlink, chmod, etc.) The dynamic linker/loader checks the environment variable LD_PRELOAD for a list of shared libraries to be loaded and searched before all others. These libraries are called *interposed libraries* because any function that is defined in one of these libraries intercepts the call to standard libraries.

We use a custom interposition library to intercept file-related library calls so that we can track changes to files during a shell session. When we intercept the call, we are able to gather information about the current state of the file before passing the request on to the real library call. We are also able to observe the resulting changes and collect additional information as needed.

We decided to use an interposition library for a number of reasons. It can be written so that it only captures the events we are concerned with, including privileged operations (using sudo, su, etc.). It is a userspace solution, so it is not dependent on any particular kernel and is portable to most other UNIX platforms. Finally and perhaps most importantly, the majority of the functionality was already present in a

⁴<http://www.unionfs.org/>

⁵<http://slax.linux-live.org/>

⁶<http://knoppix.org/>

⁷<http://fuse.sourceforge.net/>

⁸<http://www.intersectalliance.com/projects/snare/>

⁹<http://www.nsa.gov/selinux/>

component of Audlib [4, 5] designed to track user and sysadmin actions for detecting abuse of authority.

Trackle Architecture

Trackle consists of four functional components, each of which communicates with a central database:

1. The web interface, which allows full access to the ticket database, Wiki pages, and shell session logs. It is the primary interface to all Trackle data.
2. The console tools, which allow sysadmins to access the ticket database, and begin shell tracking sessions to capture changes made to resolve a problem.
3. The interposition library, which is responsible for tracking changes to files during a tracked shell session.
4. The email notification system, which keeps end users and sysadmins informed of ticket status changes, and the email confirmation system which enables unauthenticated end users to interact with aspects of the web interface.

The web interface, console tools, and email system are all written in Python and communicate directly with the central PostgreSQL¹⁰ database. The

¹⁰<http://www.postgresql.org/>

shell tracking session is written in C, and communicates through the console tools (see Figure 1).

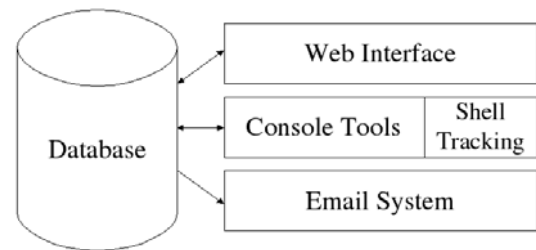


Figure 1: Trackle consists of four components which store their data in a central PostgreSQL database.

Web Interface

The Trackle web interface provides full access to tickets, shell logs, and an integrated Wiki. The web interface has been designed with simplicity and integration in mind. Like most of Trackle, the web interface is written mostly in Python. Trackle uses Clearsilver¹¹ for its HTML templates. The web interface recognizes two classes of users, anonymous end users and authenticated sysadmins. Because they are not required to log in, end users are required to complete email confirmation when creating tickets. Sysadmins

¹¹<http://www.clearsilver.net/>

The image shows a screenshot of a web browser displaying the 'Create New Ticket' form in the Trackle application. The browser's address bar shows 'http://senegal.sccs:8080/trackle/newticket'. The form has a header with 'Trackle.' and navigation links like 'Wiki', 'View Tickets', 'New Ticket', and 'Search'. The form fields include: 'Contact email address' (text input), 'Brief problem summary' (text input), 'Type' (dropdown menu set to 'defect'), 'Full description (you may use WikiFormatting here):' (rich text editor with bold, italic, and link buttons), and 'User Severity' (dropdown menu set to 'medium'). A 'Preview' button and a 'Submit ticket' button are at the bottom. A note at the bottom right says 'Note: See TracTickets for help on using tickets.'

Figure 2: The ticket creation form for anonymous users is streamlined so that a user can file a ticket quickly. The set of fields displayed and the explanation shown next to each is configurable.

will be able to authenticate with the system, which will allow them to perform privileged tasks.

Tickets: Trackle's integrated issue tracking system is more than just a to-do list. It provides the initial framework for organizing automatically collected data, linking an abstract description of a problem and the concrete steps required to solve it. All users may create tickets. End users are presented with a streamlined form (Figure 2) requesting:

- contact email address
- brief problem description
- problem type
- problem severity

In addition to these fields, authenticated sysadmins are prompted for more information which would not be relevant to end users:

- priority
- sysadmin assignment
- keywords
- subscription list

Ticket browsing is also available to anonymous and authenticated users. Again, the two user groups have very different views. Anonymous users are presented with a minimal amount of information about

the ticket, for convenience, security, and privacy. They can also subscribe to existing tickets. Authenticated users have access to the entire ticket history, have the ability to change the ticket fields, and can close, reopen, and assign tickets.

Wiki pages: Trac's integrated Wiki required very little modification for Trackle. A Wiki enables users to add, remove, and modify content easily and quickly directly through their web browser using a simple formatting language. Wiki pages are visible to both authenticated and anonymous users, but are only modifiable by authenticated users and can be made private. Most long text fields in Trackle support Wiki formatting to allow even more possibilities for cross-referencing and annotation.

Shell session logs: All the information recorded in a shell tracking session is presented in the web interface in a shell session log (Figure 3). They are accessible to authenticated sysadmins through an index where they are sorted by ticket, with the most recently added shell log appearing first. The logs are also linked individually from the associated ticket. In keeping with the ideal of providing the most functionality while imposing the least structure, Trackle shell

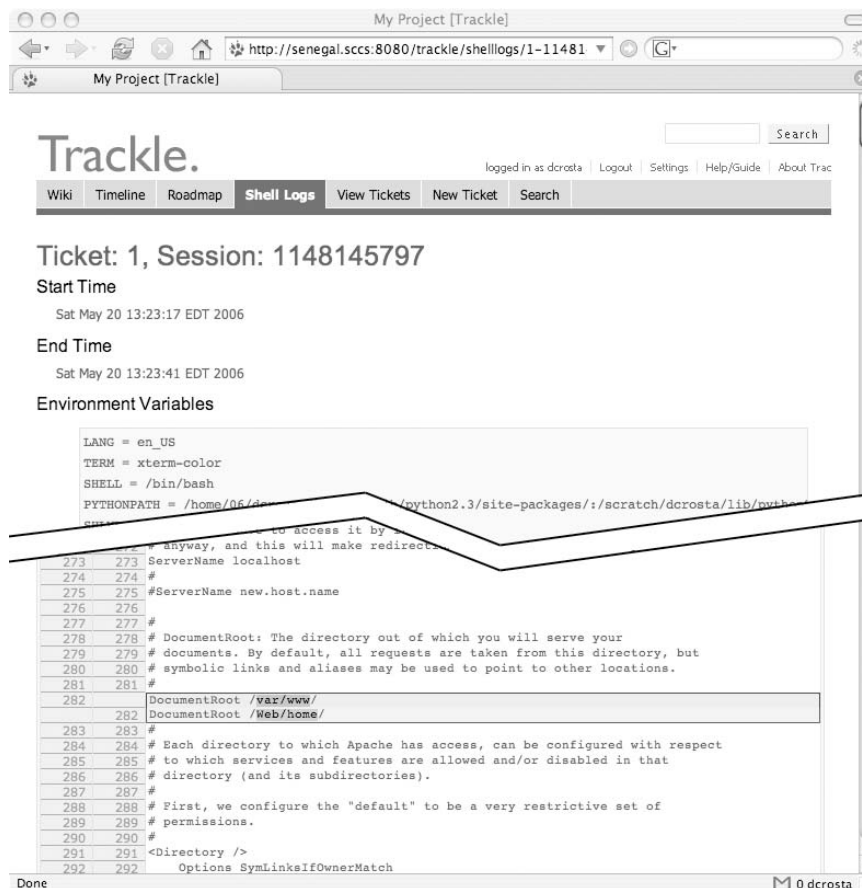


Figure 3: The shell session log displays all the data that was recorded during the shell session. At the top, the shell history, start and end times, and environment variables are displayed. Below, the contents of files that were changed are shown with deletions highlighted in red and additions in green.

logs are editable and support Wiki formatting. The Wiki-like nature of the shell logs allows for easy annotation and cross-referencing.

After the Wiki portion of the log (which contains shell history, environment variables, and any other pertinent shell data), the modified files are displayed as colored diffs. All of the modified files are stored in the database, but not all are displayed by default. The shell log edit page provides a list of all files from which individual files can be selected for display.

Console Tools

The Trackle console tools consist of several scripts supporting the main interface, `trackle-cli`. Like most of the rest of Trackle, the console tools are written in Python for easy extensibility and maintenance.

Trackle-cli: The primary interface to the ticket database from the console is `trackle-cli`, a screen-oriented program for the `curses` environment. It is written using `DTK`,¹² a Pythonic wrapper for the `curses` Python module. Following our goal of minimalistic interfaces, `trackle-cli` is designed to give enough information that sysadmins can quickly find a ticket and begin working on it, but not so much that they are bogged down by text-filled screens of details. The two primary screens of `trackle-cli` are the ticket overview and ticket detail screens.

The ticket overview screen (Figure 4) lists the open tickets, showing for each the values of a configurable set of ticket fields. The default set shows the ticket's numeric ID, summary, owner, and time of last change. Sysadmins navigate this list using the keyboard and may either select an existing ticket or create a new ticket.

¹²<https://firefly.student.swarthmore.edu/trac/wiki/DTK>

The ticket detail screen (Figure 5) shows the most pertinent details of an individual ticket, with short ticket fields displayed above the fold and the longer ticket description shown below in unformatted Wiki text. Some ticket information, notably the ticket change log, is not available through the console interface due to space constraints.

`Trackle-cli` supports editing of all visible information in the ticket detail screen. After selecting a field, the focus is moved to either a text editing field or an enumerated selection field which displays appropriate available values. Enumerated fields can have possible values added to them through the web interface or `trackle-admin` program. Ticket editing in `trackle-cli` is designed to allow a sysadmin to correct minor errors found in a ticket – full-featured editing is available in the web interface.

Sysadmins can edit the ticket description using the editor set in the `VISUAL` or `EDITOR` environment variable. When the editor terminates, the description display area is updated with the new value. If the sysadmin wishes to save the changes back to the Trackle database, `trackle-cli` prompts for a ticket changelog message using the same method.

From the ticket detail screen, the sysadmin can start a tracked shell session. `Trackle-cli` is suspended and replaced by a shell with a modified environment. Upon completion of the shell session, the sysadmin is shown a list of all the files that were modified during the session (see Figure 6). The sysadmin then selects files to display in the web shell session log. This selection may be changed later through the web interface. After selecting the files and saving the session log, the sysadmin is returned to the ticket detail screen.

ID	Summary	Owner	Changed
6	Mail spool size warnings to users, list f	mschust1	11 Apr 06 15:13
9	set up an emulation machine in the video	abdalian	02 May 06 19:01
4	create a suggestion box/envelope for the	sccs staff	03 Apr 06 21:34
11	[private] user removal script	sccs staff	09 Apr 06 08:44
8	Color printer	sccs staff	10 Apr 06 11:43
10	Get usernames showing up on the webcam ag	sccs staff	10 Apr 06 12:07
27	hours is a pain to update	kit	04 May 06 19:16
12	fix mallard's monitors	sccs staff	09 Apr 06 16:33
14	make a web designer mailing list	kit	24 Apr 06 17:24
15	get SMUG up and running	sccs staff	10 Apr 06 12:56
18	[private] get tape backup working again	sccs staff	13 Apr 06 08:33
21	[private] create VR movies of our spaces	sccs staff	22 Apr 06 03:25
22	[private] organize the cables in the cabl	sccs staff	22 Apr 06 22:17
23	[private] investigate round cube as squir	sccs staff	22 Apr 06 22:17
24	Kiwi sound always plays through internal	sccs staff	27 Apr 06 17:23
25	[private] Use closet for camera and signo	msingle1	02 May 06 18:06

Figure 4: Trackle-cli's ticket overview list, sorted by change date. The "Help Bar" at the top lists currently active keybindings. Trackle-cli is designed to work in an 80 × 24 character window.

Trackle-shell-prompt: Our experience at the SCCS with the beta release of Trackle showed that even experienced sysadmins occasionally forgot whether they were in an ordinary shell or in a tracked shell spawned by trackle-cli. To address this, we created trackle-shell-prompt, a helper script which detects whether the user is running inside a tracked shell session by checking for certain environment variables. If so, it prepends the prompt with “Trackle” in green boldface. Trackle-shell-prompt also sets its exit code to 0 when in a shell tracking session, or 1 when not, so that it can be used in shell scripts.

Trackle-admin: This tool is used to configure the run-time settings of Trackle stored in the central database. Trackle-admin is used to add, update, or remove accounts (only sysadmins need an account for Trackle – end users may use the web interface without authentication), and the following enumerated ticket attribute fields: component (a brief description of the area of the system affected by a problem), milestone, priority, user severity, and ticket type. Wiki pages can be imported from or exported to plain text files. Like trackle-shell-prompt, the exit code returned by trackle-admin is set to 0 on success, or another value on error.

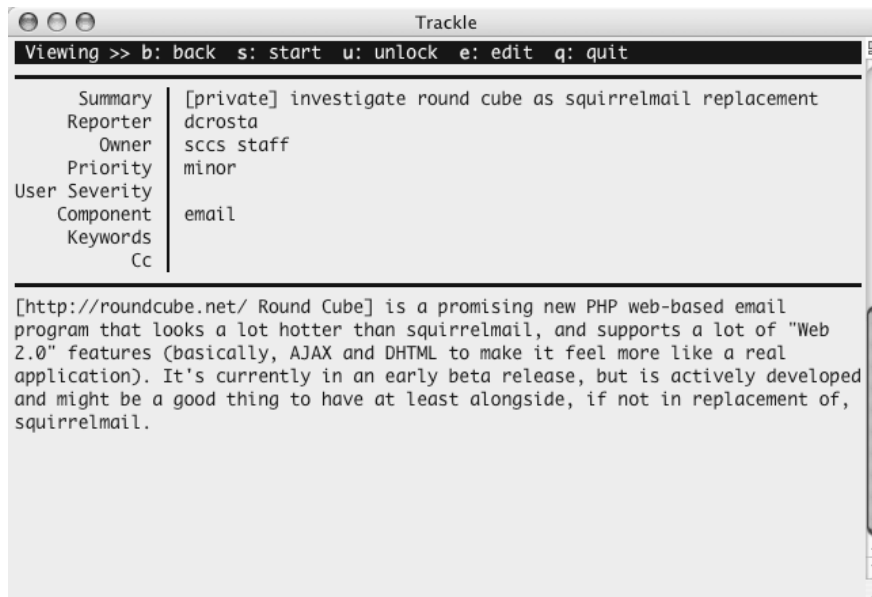


Figure 5: Trackle-cli displays the most pertinent ticket details, and allows simple editing to correct any mistakes. From this screen the sysadmin may begin recording shell actions to be associated with this ticket.



Figure 6: Trackle-cli allows the sysadmin to decide which files will be displayed in the web version of the shell session log.

Shell Session Tracking

The ability to track shell sessions sets Trackle apart from previous solutions. The tracking system removes the burden of self-reporting from the sysadmin. Relying on the sysadmin to report exactly what changes were made can be problematic. After a long session, even the most conscientious sysadmin may be unable to remember exactly what they did, let alone record it accurately. A seemingly inconsequential change made early on may be overlooked after tackling a more frustrating issue.

Recording incorrect or incomplete information defeats the purpose of tracking changes because it reduces the accuracy of the report and degrades the utility of the system as a training tool. The goal of Trackle is to automate as much record collection as possible in order to avoid human error.

To provide an accurate summary of a shell session, Trackle collects information about the session:

- start and end time
- environment variables
- shell history
- copies of all created, removed, or modified files

Most of these items are easy to collect using built-in Python commands. The history of shell commands can be obtained from the bash shell.¹³ This information is stored in a state directory created for each tracked shell session.

Recording file changes is the tricky part. For this we use library interposition. Our interposition library, *libtrackle*, is based on the work done by Kuperman and Paksoy for Audlib. We can intercept library calls by writing our own version of the call in question with the same signature. Effectively, we wrap the library call with additional functionality to support Trackle. In this function we record any information we need and then pass the call on. This whole process is transparent to the running program, and as long as the parameters and return values are passed through to the real version without modification, interposition does not change the program's behavior.

In order to catch file modifications we need to intercept file-related calls (*fopen*, *chmod*, etc.) before they get to the kernel so that we can make an initial copy of the file in question. Once we have a copy, we allow all further calls on that file to pass through to the kernel without logging. Just before the program terminates, we make another copy of the file to compare against the original version.

In addition to intercepting calls, interposition allows us to define functions that are called when the library is first loaded and when the process terminates. We use the initialization routine to cache local state variables originally set in the environment by the

¹³Trackle currently only supports the bash shell. Support for *tcsh* is planned for a future release.

console tools. In the finalization function we iterate over all files that have been accessed during the execution of the program and make a final copy of all files which have been modified.

A file might be modified several times during the course of a tracked shell session. We record the initial copy only once but overwrite the final copy each time a program terminates. Therefore, the difference information generated consists of changes made over the entire session.

Some programs written with security in mind attempt to disable library interposition (for good reasons), and sysadmins are encouraged to use these sorts of tools (e.g., *sudo*) when changing system configuration. In order to collect these changes, we need to prevent these programs from disabling *libtrackle*. For example, the environment created for executing *vim* by "*sudo vim /etc/filename.cfg*" lacks the *LD_PRELOAD* environment variable, effectively disabling *libtrackle*. We could not find a way to override this behavior in *sudo* without modifying its source code. We circumvented this issue by overriding the *exec* library calls with our own versions that reset the *LD_PRELOAD* variable to the proper value before executing the program.¹⁴ This solution ensures that *libtrackle* is loaded for all dynamically linked programs.

Email Notification and Confirmation

Email plays two roles in Trackle, providing notification to sysadmins and end users, and email confirmation to prevent abuse of the web interface by unauthenticated users.

Whenever a sysadmin changes a ticket's status, like closing an open ticket or assigning a new ticket, a configurable list of staff email addresses (in SCCS's case, our staff email list address) is sent a notification indicating the status change and any comments or changed fields changed at the same time. Subscribed users receive an abbreviated version of this email containing just the status change information and the comment.

Trackle also provides the *trackle-notify* script which sends periodic updates to the staff email addresses. The email contains a list of all open tickets, details for each open ticket, and a list of ticket status changes since the previous periodic update was sent. *Trackle-notify* is designed to be run through *cron* or another task scheduler, therefore it produces no output under normal circumstances.

The email confirmation system allows end users to confirm their identity without requiring them to get, remember, and use a separate set of user names and passwords. When an end user creates a ticket or subscribes to an existing ticket, Trackle sends an email to

¹⁴For *setuid* programs, *ld.so* only honors values of *LD_PRELOAD* that are in the default library search path and are also *setuid*. A user would have needed administrative access to put such a library in place, so this is not a major vulnerability.

the address the end user entered. Users click the link in the email to make the action take effect.

Sample Workflows

Trackle is designed with human interaction in mind, and as such any description of its components does not fully capture the feel of the application. In this section we describe two sample workflows to demonstrate the expected usage of Trackle: 1) the lifecycle of a typical ticket created by an end user, and 2) a sysadmin annotating existing resources for future reference.

Ticket Lifecycle

Suppose SCCS user Alice is unable to access the administration page for her group's email list. She points her browser at Trackle's ticket creation page, <http://bugs.sccs.swarthmore.edu/>, and begins to fill out the form. She is prompted for her email address, a brief summary, a ticket type (software bug, software request, security issue, etc.), a user severity (that is, how severe the issue is to Alice) and a longer description of the problem. Alice notices the link to a reference on Wiki formatting, and helpfully formats her description into several sections including error output she received, a link to the page with the problem, etc.

When Alice clicks the submit button, she is taken to a screen informing her that she will receive an email containing instructions and a link to confirm her ticket, and that her email address has been automatically subscribed to the ticket for status updates. Without clicking this link, her ticket will not show up in ticket reports, and the sysadmins will not be notified. Once she clicks the link, Trackle marks the ticket as confirmed and emails the sysadmins to notify them of the newly created ticket.

Now suppose SCCS user Bob is also unable to access the administration page for his group's email list. Because he read the page on ticket etiquette, he knows to first use Trackle's built-in search feature to see if any open tickets are already filed for this problem. He notices the ticket Alice just created, and confirms that it is for the same issue he is facing, so he does not need to file his own ticket. But Bob is impatient and has important work to do on his list, and wants to know as soon as the problem is resolved. Fortunately, Trackle allows Bob to subscribe to the ticket Alice created, again using email confirmation to verify his identity.

Now suppose SCCS sysadmin Melvin checks either his email or the Trackle website and notices the newly created ticket. Being our resident expert on Mailman list administration, Melvin SSH'es to our primary web and email server, runs `trackle-cli`, and begins working on the ticket. Behind the scenes, `trackle-cli` has locked his ticket so that no other sysadmin can begin working on the same issue. His shell history as well as any files he changes are now being recorded, so as he investigates and solves the problem, all his actions are

logged. When he types 'exit' to leave the shell tracking session and returns to `trackle-cli`, he is shown a list of all the files whose contents or permissions he changed, and he can select which will be included in the log by default. When he saves the shell session log to the database, the ticket is automatically unlocked.

Since, by virtue of his Mailman wizardry, Melvin resolved the problem for Alice and Bob, he opens up the ticket's page in Trackle's web interface, makes some remarks to be appended to the ticket's history, and marks the ticket closed. Trackle then emails all the subscribed users to notify them that the issue has been resolved. This notification includes the remarks Melvin appended to the ticket's history.

Annotation and Cross-Referencing

Suppose SCCS sysadmin Wendy, our resident Wiki enthusiast, decides it is time for her to learn more about Mailman. She begins by searching through the existing Wiki pages, tickets and shell session logs to find anything that might be related to Mailman lists. She creates a new Wiki page, which she protects from anonymous access since it may contain sensitive information that should not be leaked to the public, and begins adding links.

In Trackle, all Wiki pages, tickets, shell session logs, and milestones are linkable objects, and all have built-in Wiki support so that any of them can link to any of the others. Wendy takes advantage of this by not only including forward links from the new Wiki page to the related tickets and shell session logs, but by editing those objects to include return links to the new page she is creating.

By default, the shell session logs that are captured by Trackle are pretty bare and contain just collected information. Since Wendy is in no hurry, she is at ease to spend some time investigating the effects of the commands recorded in the shell history of the session log created by Melvin above. She can integrate this information, along with links to other Trackle objects, online documentation, or any website, by editing the Wiki page at work behind each shell session log. She can remove commands from the history that are not pertinent so that the shell session log contains just the relevant information for future reference. She can also change the set of files which are displayed in the shell session log, overriding the defaults Melvin selected at the end of his shell tracking session.

Future Plans

As with any large software project, the initial public release of Trackle is far from complete. As frequently happens with new software, some of the most interesting features were not suggested until the SCCS started using Trackle, and others were inspired as a byproduct of the development process. Many of these new feature ideas that came to us mid-project made it in to the initial public release, but some did not. The

following are planned features for Trackle that have not yet been implemented.

Ticket extensions: Early feedback from SCCS sysadmins has shown that some extensions to the ticket system may be useful, particularly the ability to express relationships among active tickets. A dependency relationship (the completion of ticket B can only happen after ticket A is closed) could hide or deprioritize tickets depending on others, or emphasize tickets which are depended on. A parent-child relationship would be used to group several related tickets (e.g., configuration upgrades) under one parent (e.g., Linux distribution upgrades). Ticket due dates would enable Trackle to automatically escalate a ticket's importance over time. Finally, private tickets (not visible to unauthenticated end users) would be used to track sensitive information such as hiring or policy debates.

Multiple machine support: All of the components of Trackle interact with one central PostgreSQL database. Currently, the Trackle console tools do not keep track of host-specific info, so only one machine can be tracked per instance of Trackle. Because PostgreSQL communicates over TCP, it should not be difficult to add network functionality to the console tools. It would then be possible to install the console tools on any number of properly configured servers and clients, and have them all report back to one central database.

There are, however, some situations where a per-machine instances of Trackle might be useful. Trying to track many disparate issues occurring on unrelated machines would become cumbersome and is unnecessary. An alternative that might provide the best of both worlds would be a hierarchical approach. Rather than storing all the data on one central server, leave the data distributed, but allow communication between the individual instances of Trackle. For example, deploy one network-wide overview server, one site-wide overview server per site, and install Trackle individually on the machines at each site. This could help mitigate some of the scaling issues that would come with very large databases.

File revision control: We briefly mentioned configuration management tools, some of which include revision control functionality. Trackle is based on the open source program Trac, which includes tight integration with the Subversion revision control system, and could provide revision control for all files touched during shell session tracking. A straightforward approach of storing one revision per session would not work, as the files in question may also change between tracked shell sessions. A better approach is to create a revision at each of the before and after states for each file in the repository.

Further high-level abstractions: One recent innovation of Web 2.0 technologies is the establishment of new interface and organization paradigms that more closely model how people think about information. In particular, the tagging concept, where arbitrary words

or phrases are associated with each idea or object in a system, supported by an interface which makes suggestions about tags to apply, would further increase the utility and scalability of Trackle. Some tags for shell session logs could be automatically generated, for instance, a tag for each file and each directory involved in a particular shell tracking session. Tags could also be assigned for software packages involved in a shell tracking session, by integrating with package management tools (dpkg, rpm, etc.).

Conclusions

Our experience with the SCCS staff email list and Wiki has shown that relying on self-reporting leads to missing, incomplete, or inaccurate reports of changes made to our systems. The poor quality of these reports makes it hard to find the source of a particular change. Further, an inaccurate report might cause a sysadmin trying to duplicate past steps to cause new errors instead of repairing existing ones. Additionally, the presence of inaccuracies degrades confidence in all reports. Trackle alleviates these problems by keeping consistent and accurate records so that sysadmins can focus on solving problems rather than the tedious task of keeping logs.

Trackle's integrated Wiki has allowed us to begin collecting related topics into a sysadmin training manual and how-to guide. This evolving guide allows new sysadmins to ask more sophisticated questions of their experienced colleagues. Trackle allows sysadmins to learn on their own time and at their own pace so that no one gets bored or left behind.

Often, volunteer sysadmins learn the intricacies of a system only when it breaks. Trackle allows us to learn by reflection rather than by struggling to fix a critical error. This leads to more efficient use of office time for those not present when problems occur. We also learn from authentic situations rather than toy problems or contrived examples.

Because our shell tracking tools operate transparently, Trackle can be used to complement existing change/configuration management systems. Though many configuration management systems have already solved the problem of discovering file changes, Trackle goes further to associate file changes with a particular issue. Additionally, Trackle detects changes to any files, not just those that are already being monitored by a configuration management system.

Availability

Trackle is open source software, licensed under the BSD license. You may download stable Trackle releases and documentation from <http://www.sccs.swarthmore.edu/org/trackle/>.

Acknowledgements

We would like to thank Benjamin A. Kuperman and Mustafa Paksoy for their work on Audlib [5], on which

libtrackle is based. We would also like to thank Edgewall Software and the numerous contributors to Trac, without which Trackle would not have been possible.

Author Biographies

Daniel S. Crosta graduated from Swarthmore College in June, 2006, with a B.A. in Computer Science. During his tenure at Swarthmore, he served three years as Systems Administrator for the Swarthmore College Computer Society, most recently as Lead Systems Administrator. He has also participated in research in Computer Graphics at Princeton University, and in Computer Vision at Swarthmore College. Since July, 2006, he has been working as a Software Developer at Wireless Generation in New York City. Contact him electronically at dcrosta@scs.swarthmore.edu.

Matthew J. Singleton is a currently a senior at Swarthmore College in Swarthmore, PA, where he is a double-major in Computer Science and Linguistics. He is also the Lead Systems Administrator for the Swarthmore College Computer Society. He has participated in Computational Linguistics research in the Department of Computer Science at Swarthmore College. Reach him electronically at msingle1@scs.swarthmore.edu.

Benjamin A. Kuperman received the M.S. and Ph.D. degrees from the Department of Computer Sciences at Purdue University in 1999 and 2004. He is an assistant professor at Oberlin College in Ohio and previously taught at Swarthmore College in Pennsylvania. While at Purdue, he was a researcher in the Center for Education and Research in Information Assurance and Security (CERIAS) for five years and was affiliated with COAST before that. His main areas of research are on host-based computer security monitoring systems and OS level audit systems. Reach him electronically at Benjamin.Kuperman@oberlin.edu.

Bibliography

- [1] Anderson, Paul and Edmund Smith, "Configuration Tools: Working Together," *Proceedings of LISA 2005: 19th Systems Administration Conference*, pp. 31-37, December, 2005.
- [2] *Common Criteria for Information Technology Security Evaluation*, <http://www.commoncriteria.org/>.
- [3] Evard, Rémy, "An Analysis of UNIX System Configuration," *Proceedings of LISA 1997: 11th Systems Administration Conference*, pp. 179-193, October, 1997.
- [4] Kuperman, Benjamin A., *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources*, Ph.D. thesis, Purdue University, West Lafayette, IN, August, CERIAS TR 2004-26, 2004.
- [5] Paksoy, Mustafa and Benjamin A. Kuperman, "Audlib: Generating computer security audit logs with interposing libraries," Presented at 2005 Swarthmore College Sigma Xi poster session, September, 2005.
- [6] Pomeranz., Hal "PLOD: Keep Track of What You're Doing," *Proceedings of LISA 1993: 5th Systems Administration Conference*, November 1993.
- [7] Roth, Mark D., "Preventing Wheel Reinvention: The psgconf System Configuration Framework," *Proceedings of LISA 2003: 17th Systems Administration Conference*, pp. 205-211, October, 2003.
- [8] Sharp, James M., "Request: A Tool for Training New Sys Admins and Managing Old Ones," *Proceedings of LISA 1992: 4th Systems Administration Conference*, October, 1992.
- [9] Stepleton, Thomas, "Work-Augmented Laziness with the LOS Task Request System," *Proceedings of LISA 2002: 16th Systems Administration Conference*, November, 2002.
- [10] US Department of Defense, *Trusted Computer Systems Evaluation Criteria* (also known as the 'Orange Book') Technical Report DoD 5200.28-STD, DoD Computer Security Center, Fort Meade, MD, December, 1985.

THE USENIX ASSOCIATION

Since 1975, the USENIX Association has brought together the community of system administrators, developers, programmers, and engineers working on the cutting edge of the computing world. USENIX conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems. USENIX and its members are dedicated to:

- problem-solving with a practical bias
- fostering technical excellence and innovation
- encouraging computing outreach in the community at large
- providing a neutral forum for the discussion of critical issues

Membership Benefits

- Free subscription to *login:*, the Association's magazine, both in print and online
- Online access to all Conference Proceedings from 1993 to the present
- Access to the USENIX Jobs Board: Perfect for those who are looking for work or are looking to hire from the talented pool of USENIX members
- The right to vote in USENIX Association elections
- Discounts on technical sessions registration fees for all USENIX-sponsored and co-sponsored events
- Discounts on purchasing printed Proceedings, CD-ROMs, and other Association publications
- Discounts on industry-related publications: see <http://www.usenix.org/membership/specialdisc.html>

For more information about membership, conferences, or publications, see <http://www.usenix.org>.

SAGE, a USENIX Special Interest Group

SAGE is a Special Interest Group of the USENIX Association. Its goal is to serve the system administration community by:

- Establishing standards of professional excellence and recognizing those who attain them
- Promoting activities that advance the state of the art or the community
- Providing tools, information, and services to assist system administrators and their organizations
- Offering conferences and training to enhance the technical and managerial capabilities of members of the profession

Find out more about SAGE at <http://www.sage.org>.

Thanks to USENIX & SAGE Supporting Members

Addison-Wesley Professional/
Prentice Hall Professional
Ajava Systems, Inc.
AMD
Cambridge Computer
Services, Inc.
cPacket Networks
DigiCert® SSL Certification
EAGLE Software, Inc.
Electronic Frontier Foundation
FOTO SEARCH Stock Footage
and Stock Photography

GroundWork Open Source
Solutions
Hewlett-Packard
IBM
Infosys
Intel
Interhack
Microsoft Research
MSB Associates
NetApp
Oracle

OSDL
Raytheon
Ripe NCC
Sendmail, Inc.
Splunk
Sun Microsystems, Inc.
Taos
Tellme Networks
UUNET Technologies, Inc.
VMware

ISBN-13: 978-1931971485
ISBN-10: 193197148X



9 781931 971485